Proyecto Fin de Carrera

Ingeniería de Telecomunicaciones

# Síntesis de sonido interactiva para videojuegos mediante Unity y PureData


Autor

Sergio Pina Lagunas

Director

José Ramón Beltrán Blázquez

Escuela de Ingeniería y Arquitectura



Septiembre de 2014

# Agradecimientos

El viaje llega a su fin y no puedo decir que no haya disfrutado del camino. Este documento cierra una etapa de mi vida y abre otra y me gustaría recordar aquí a los que, de alguna manera, han puesto su granito de arena para no olvidarla nunca.

En primer lugar, me complace mostrar mi agradecimiento y respeto a mi tutor en el desarrollo de este Proyecto Fin de Carrera, José Ramón Beltrán Blázquez, por darme la oportunidad de realizar un trabajo estimulante, compartir conmigo sus valiosos conocimientos y experiencia y prestarme su inestimable ayuda y valioso tiempo cuando los he necesitado.

Me gustaría también en estas líneas acordarme del resto de profesorado en general, y del relacionado con la Ingeniería de Telecomunicaciones en particular. Con algunos he tenido desavenencias, con otros la relación ha sido fluida y enriquecedora, y con otros, simplemente, indiferente. Pese a ello, con los años uno se da cuenta del talento que hay en la Escuela de Ingeniería y Arquitectura y lo mucho y duro que han tenido que pelear para alcanzar sus puestos. Mi respeto y admiración también va para ellos.

Mención especial merecen los soldados que me han acompañado en esta guerra. Los que ya están colocados, incluso fuera de casa, los que están intentando colocarse, y los que todavía les falta finiquitar el asunto. Compañeros de buenos y malos momentos, de juergas, de viajes y de horas de biblioteca. ¡Gracias a todos, huevazos!

Tampoco puedo dejar de recordar a las amistades que me acompañan desde que apenas sabía poner una palabra detrás de otra con sentido. Sin los buenos momentos que me sacaban de la rutina y sin el apoyo y desconexión que me proporcionaban, el camino habría sido mucho más largo y duro o, quién sabe, tal vez no habría llegado a buen puerto.

Gracias a mis amigos de más allá de los Pirineos, aunque no puedan entender estas palabras. Nunca podré olvidar las experiencias vividas durante el curso 2012/2013 en Göteborg, a toda la gente y los buenos momentos. Nos vemos en la próxima.

Gracias a Elena, por, de una forma u otra, estar siempre ahí, especialmente en las duras. Gracias por tener siempre la palabra amable que necesito y el abrazo reconfortante. Eres la mejor.

A mi abuelo, cuya única preocupación que prácticamente le quedaba en la vida era llegar sano a este día y que lleva meses con el traje preparado.

Y por último, solo me queda acordarme de las dos personas con las que empezó todo. Nunca podré compensar a mis padres lo que han hecho por mí, nunca podré agradecerles lo suficiente los esfuerzos y sacrificios que han hecho para sacarme adelante. Infinitas gracias por el apoyo, por la educación, por poner los cimientos de la persona que soy, por inculcarme los valores de la honestidad y el esfuerzo, por aguantarme en mis momentos malos, por apoyarme en los momentos más bajos. Sois los principales responsables de que esté hoy aquí.

¡Muchas gracias a todos!

# Resumen

El audio procedural es una técnica que puede ser empleada, entre otras cosas, para la programación de videojuegos y, aunque ha sido utilizada en diferentes trabajos, no ha sido desarrollada del mismo modo que las técnicas de síntesis y generación de imágenes.

En este proyecto, se ha realizado una inmersión en la disciplina del audio procedural y sintético, intentando comprender las técnicas y las características propias de la materia, y utilizarlas para implementar una serie de efectos de sonido que se puedan aplicar a diferentes escenas interactivas mediante el software de programación gráfica Pure Data.

Para dar soporte a estos efectos implementados, se ha recreado gráficamente un entorno montañoso en un entorno de programación de videojuegos denominado Unity, en el cual un personaje principal interactúa con el ambiente, provocando eventualmente con sus acciones la reproducción de estos efectos de sonido.

Posteriormente, se ha utilizado el protocolo OSC para comunicar ambas plataformas. El “feedback” creado por la interacción del usuario por medio del personaje principal en la escena es traducido a mensajes OSC e interpretado con Pure Data para reproducir de manera adecuada los efectos de sonido.

Finalmente, con la experiencia obtenida en el desarrollo del trabajo, se han extraído ciertas conclusiones sobre el papel del audio procedural en un futuro en el mundo de los efectos de sonido.

# Índice general

# Índice de figuras

# Capítulo 1: Introducción

## 1.1 ¿Qué es el Audio Procedural?

El Audio Procedural consiste en la creación de sonidos sintéticos que se generan exclusivamente mediante algoritmos y que pueden ser modificados en tiempo real mediante un conjunto de parámetros de entrada. La síntesis de dichos sonidos se puede realizar mediante diferentes técnicas, cada una de ellas adaptada al tipo de sonido asociado a la realidad sonora que se quiere representar.

El Audio Procedural se enmarca dentro de la generación sintética de sonidos. Las tecnologías que permiten la reproducción de sonidos se pueden clasificar en: audio lineal, grabado, interactivo, adaptativo y secuenciado, sintético, generativo y artificialmente inteligente. Otros términos de cara a la creación musical son la composición algorítmica o música estocástica [1].

*-Sonido grabado*

La tecnología tradicional del audio tiene sus cimientos en la grabación. Señales de sonido del mundo real se capturan mediante un micrófono, mezcladas, procesadas y, finalmente masterizadas de manera definitiva. Cada vez que se vuelve a reproducir, no cambia, la pieza queda exactamente igual que después de ser masterizada. Por el contrario, una composición de audio procedural puede cambiar sus características cada vez que se reproduce. El sonido grabado consiste en datos donde los valores marcados en una secuencia de tiempo corresponden a amplitudes, normalmente 44100 por segundo. Las muestras son reproducidas de inicio a fin en el mismo orden y al mismo ritmo en el que fueron grabadas. Esto es conocido como muestreo, una tecnología que ha estado establecida desde hace décadas. Así pues, los dispositivos reproductores convierten la información que reciben de nuevo en sonido.

Por su parte, el audio procedural no necesita almacenar ningún tipo de información más allá del software donde está implementado. El concepto de audio procedural, de hecho, se reduce a dicho software. El programa es la música o el sonido, implícitamente contiene la información y los medios para crear audio.

*-Sonido interactivo, no lineal y adaptativo*

Así como un programa puede aceptar entradas, es posible modificar el sonido de salida durante la reproducción. Esto es conocido como audio interactivo. Cuando reproducimos un CD o un DVD, no necesitamos realizar más acciones hasta que el disco termina. El orden de la información está fijado y el progreso de un valor hacia el siguiente es automático. Algo que empieza en un punto y se mueve a ritmo constante a otro es considerado lineal desde el punto de vista temporal. Alternativamente, audio interactivo utiliza las entradas que le son provistas para crear nuevos sonidos o modificar los existentes, y el valor de la entrada afecta a la al sonido final producido. Un piano MIDI es un buen ejemplo de sonido interactivo simple. Para

escuchar una nota se debe presionar una tecla y la intensidad del timbre y de la nota se determina por la velocidad de la tecla, y la cantidad de presión ejercida sobre ella. Evidentemente, un piano consiste en más de un sonido diferente, concretamente hasta 88, uno por tecla. Algo que puede saltar entre un número de valores en cualquier orden (discontinuamente), o moverse a diferentes ritmos en diferentes direcciones es no lineal en el tiempo.

A día de hoy, todos los sonidos de videojuegos pueden ser considerados como audio interactivo (salvo las piezas de música de fondo en algunos casos, que son lineales). Cada sonido que se escucha depende de la acción de un jugador o de un evento del mundo virtual. El orden y la temporización de estos sonidos no está predeterminado como con el audio grabado. Las aplicaciones interactivas, como los videojuegos, pueden incorporar relaciones altamente elaboradas entre las entradas generadas por el usuario y el sonido de salida, pero el principio común que fundamenta el audio interactivo es la necesidad de disponer de un conjunto entradas por parte del usuario. En un videojuego, en ciertas situaciones se cambia la música o los efectos de sonido para provocar cambios emocionales en el jugador. Esto es lo que se conoce como audio adaptativo, es una forma de sonido interactivo donde una función compleja o máquina de estados se interpone entre las acciones del jugador y la respuesta audible [1].

*-Sonido secuenciado*

Entre el sonido interactivo y grabado se encuentra lo que normalmente es conocido como sonido secuenciado. Este el método que utiliza la mayoría de la producción musical hoy en día para géneros como el hiphop, el pop o el rock. Estos sonidos son pequeños clips grabados de instrumentos individuales o líneas vocales que son almacenadas en un secuenciador. El secuenciador actúa como una herramienta para el compositor para recomponer las partes pregrabadas y entonces reproducirlas en un orden fijado. El sonido secuenciado ha sido utilizado en videojuegos durante mucho tiempo, pero ha caído en el desuso [1].

*-Sonido sintético*

Los sonidos sintéticos son producidos por hardware electrónico o simulaciones digitales en hardware de osciladores y filtros. El sonido se crea directamente desde la nada, utilizando ecuaciones que expresan funciones en el tiempo. No se necesitan más datos. Los sintetizadores producen formas de onda de sonido con una forma dinámica, espectro y amplitud característicos. Pueden ser generados con el objetivo de simular instrumentos reales como el piano, o no reales. La combinación de secuenciadores y sintetizadores es la base de la composición en géneros musicales como el techno o el dance, además de ser propicios para realizar sonidos o música ambientales. Los sintetizadores también juegan un rol no musical, pudiendo recrear efectos de sonido naturales como el viento, la lluvia, truenos etc [1].

*-Sonido generativo*

Sonido generativo es un término complejo, que engloba otros como algorítmico, procedural y artificialmente inteligente. En este solapamiento, frecuentemente, el concepto real que se menciona es el mismo, se hable de uno o de otro. Sonido generativo es aquél generado por algún proceso o conjunto de procesos, en oposición a la composición por parte de una persona. Habitualmente, el sonido generativo no necesita entradas, o dichas entradas se dan como ciertas condiciones iniciales previas a la ejecución. Sin embargo, analizando algunas implementaciones prácticas, esta definición no se sostiene, pero ciñéndonos a ella, generativo no implica interactivo. Las composiciones generativas difieren de las secuenciadas en que las segundas no cambian y están previamente programadas, mientras que las primeras suceden durante la ejecución del programa que las controla. Son determinísticas, y no completamente aleatorias, porque se ejecutan en un ordenador, pero no predecibles por la percepción humana o la razón [1].

*-Sonido estocástico*

Algunas formas de generar sonido, normalmente referidas a sistemas estocásticos, utilizan datos aleatorios o caóticos. Filtran estos datos aleatorios para obtener el orden a través de las reglas matemáticas de la estadística y la distribución. Alternativamente pueden utilizar métodos algorítmicos para generar datos prácticamente aleatorios con un alto grado de complejidad pero con una distribución bien definida. En la mayoría de los sistemas estocásticos, la parte generativa es puro ruido blanco y la distribución es determinada por cómo es filtrado.
El sonido estocástico puede ser generativo o interactivo ya que las entradas del usuario pueden ser parámetros de la ecuación generadora o parámetros para filtros que operan con la información generada [1].

*-Sonido algorítmico*

La composición algorítmica se refiere normalmente a un proceso que evoluciona de acuerdo a un conjunto de reglas simples y fáciles de entender. Un algoritmo se define como un conjunto de reglas para resolver un problema en un número finito de pasos. Una clase de sonidos generativos son conocidos como los métodos matemáticos que se encargan de encontrar secuencias con propiedades musicales útiles mediante métodos iterativos. En este tipo de música algorítmica, la ‘respuesta’ al ‘problema’ reside en los pasos a través de los cuales funciona el algoritmo. Lo interesante son las soluciones parciales a lo largo del camino, no el resultado final.

Un importante punto discordante entre sonido sintético y algorítmico es que la síntesis trata normalmente con sonidos producidos al nivel de la muestra, de la forma de onda, bajo un cuidadoso control, mientras que el algorítmico tiende a modificar los parámetros utilizados para controlar estas formas de onda [1].

*-Sonido inteligente*

Una clase de métodos algorítmicos, más complejos que las secuencias matemáticas, son los conocidos como algoritmos de inteligencia artificial (IA). Todos los algoritmos utilizan algún tipo de memoria para almacenar variables intermedias como los últimos dos o tres valores computados, pero éstos son normalmente descartados inmediatamente para ahorrar espacio. Un algoritmo de IA es mucho más persistente, mantiene la información de estado y de entradas a lo largo de la ejecución mientras evalúa nuevas entradas. Cuando a un secuenciador generativo se le da información adicional que equivale a conocimiento y capacidad de realizar elecciones basadas en nuevas entradas, decimos que es de IA. La información de entrada es procesada por filtros y reconocedores de patrones que llaman a acciones que producen una salida en forma de audio acorde con la entrada. Típicos ejemplos son los llamados sistemas expertos, redes neuronales, algoritmos genéticos o los autómatas celulares [1].

*-Audio Procedural*

El concepto de “audio procedural” está compuesto por dos términos. Una definición para el adjetivo procedural es aquello que está relacionado con la manipulación de símbolos, conceptos y reglas para completar una tarea o solucionar un problema. Dicho problema se refiere a producir sonido adecuado a una o más restricciones. En definitiva, audio procedural es aquél que depende de ciertos parámetros variables, como una magnitud física, u otro parámetro producido durante una partida en un videojuego para producir una salida que, bien es directamente audio, o son parámetros para controlar una posible salida de audio [1].

Conforme la tecnología ha avanzado ha surgido un problema, el de cómo proveer a las colosales cantidades de contenido requeridas para poblar los mundos virtuales de los videojuegos modernos con efectos de sonido. Mientras la rama dedicada a la apariencia gráfica ha recibido una gran inversión monetaria y atención, la rama del audio no ha sido tan bien cuidada [28].

*-Ventajas*

Las ventajas del audio procedural son que requiere un manejo sobre el código para recrear las condiciones bajo las que se tiene que reproducir cierto sonido, en lugar de tener grabaciones relacionadas con eventos, lo que produce una alta ocupación de memoria. Esto hace que haya posibilidades de incluir jerarquías en las cuales haya códigos que deriven de otros más generales, lo que hace los sistemas, aunque más complejos, más compactos.
En el caso de audio grabado previamente, cada muestra debe ser traída desde un almacenamiento secundario a la memoria RAM o, directamente por una tubería al núcleo durante la reproducción, lo que significa que sobrecarga los buses de transmisión. Sin embargo, con el audio procedural solo se ve involucrada la CPU que lo genera. Las

instrucciones necesarias para ello son muy pequeñas, lo que apenas tiene repercusión en los buses. La CPU trabaja para computar la información en tiempo real. Este cambio radical tiene importantes efectos en la arquitectura del sistema y el diseño, como el gran aumento de rendimiento en la transmisión de información para otros usos.

El anterior modelo requiere que la mayor parte del trabajo se haga por adelantado, previa a la ejecución en la plataforma. Decisiones como los niveles de sonido, el mapeado de eventos referentes al sonido, y elección de efectos son tomadas de antemano. Por otra parte, el audio procedural es altamente dinámico y flexible y aplaza las decisiones hasta el momento de la ejecución mientras se satisfaga el problema del coste de predicción. Esto significa que las elecciones críticas estéticas pueden realizarse durante el proceso final de producción, de forma similar a la postproducción de una película. El equipo así puede concentrarse en las escenas importantes y mezclar música y efectos más eficientemente.

Con el audio procedural, los productores o programadores se concentran en el comportamiento y físicas de clases enteras de sonidos, en lugar de trabajar con información recogida previamente y motores de reproducción. El diseñador pasa a trabajar más como un programador. Un objeto de sonido es realmente un método de un objeto del mundo virtual, lo que elimina la división entre los medios visuales y de audio. La ventaja es que, modificando las propiedades del objeto, se modifican también las propiedades del sonido que produce.
El audio procedural incorpora versatilidad, unicidad y un nivel de detalle dinámico. Puede ser altamente interactivo con parámetros continuos en tiempo real.

La reproducción en segundo plano de información tiene un coste fijo, no importa de qué sonido se trata, siempre requiere del mismo coste computacional. El audio procedural tiene un coste computacional variable, cuanto más complejo es el sonido, mayor cantidad de recursos necesita. Pese a ello, esto supone una gran ventaja. Aunque estéticamente, los sonidos grabados son más reales, una pequeña cantidad de éstos normalmente supera ampliamente una gran cantidad producida con las técnicas del audio procedural en cuanto a recursos consumidos. Si bien es cierto que algunos tipos de sonido son muy difíciles de recrear proceduralmente, otros, como ciertos fenómenos naturales, son más fácilmente recreables y no suponen un gran consumo computacional. Además, añadiendo técnicas de psicoacústica o métodos perceptuales para construir las partes del sonido que nos interesan en una cierta escena o eliminar las frecuencias que no tienen un aporte esencial todavía se puede optimizar más el rendimiento. El coste debido a sonidos periféricos, de esta manera, es altamente reducido [1].

*-Desventajas*

Las desventajas del audio procedural son también palpables. El audio sintético generado dinámicamente no es una panacea. De hecho, hay áreas donde nunca reemplazará al sonido pregrabado. Todavía tiene que hacer frente a algunos problemas de software, tales como cómo manejar el ajuste de fase en los métodos procedurales.

Para los efectos de sonido, el realismo es un problema a resolver que parece tener más una categoría política que técnica. Hay voces que reclaman que el audio sintético no es demasiado realista. Un alto grado realismo es posible, pero es algo que no es alcanzable en

todos los campos, como se ve a lo largo del desarrollo del proyecto, y que es inherente a la opinión de cada individuo que escuche un efecto de sonido sintético.

El coste variable puede ser también considerado como una desventaja. El coste de producir un efecto sintético es difícil de predecir antes de la ejecución y no se sabe cómo localizar los recursos. Este problema es común a otros métodos de producción de contenidos dinámicos y requiere que, o bien se haga una estimación de estos costes, o producir métodos que bajen su calidad conforme se vayan acabando los recursos en lugar de dejar de reproducirse abruptamente. Otra solución sería reestructurar programas para incluir una etapa de precomputación, pequeños tiempos de carga.
Incluso los diseñadores de sonido expertos habituados a la tecnología progresiva, se sienten incómodos con la necesidad de adaptar sus habilidades y aprender a utilizar las nuevas herramientas para generar audio procedural.

Hay una enorme industria construida alrededor del modelo que utiliza información recogida previamente en forma de librerías de muestras y el audio procedural es altamente disruptivo con ello. Un simple software basado en un modelo físico para estructuras metálicas podría reemplazar sonidos grabados de campanas, puentes de hierro, puertas, cañones de armas, llaves y otros miles de sonidos con 1KB de código. Aunque parece que el audio procedural está amenazando el rol tradicional de los diseñadores de sonido, no es así. El lugar que esta tecnología tiene que ocupar es la generación automática de sonidos para nuevos proyectos de gran tamaño. Ambos pueden y deben coexistir [1].

## 1.2 Herramientas utilizadas

### 1.2.1 Pure Data

Pure Data (en adelante, PD) es una plataforma abierta con un lenguaje de programación gráfico para Windows, Mac OS X y Linux. PD es un poderoso y rápido entorno de desarrollo para audio en comparación a C++, siendo la curva de aprendizaje para artistas como los diseñadores de sonidos de videojuegos mucho más suave. Usar PD le puede dar al diseñador control creativo adicional, además de desarrollar valiosas habilidades y vocabulario para trabajar mejor con codificadores de sonido durante la implementación de prototipos [29].

El motor de PD puede ser fácilmente integrado con otros proyectos, como el plug-in de navegador [30].

PD evita el tiempo gastado compilando cuando se hacen cambios, es posible modificar parámetros y comportamientos de objetos mientras el parche está ejecutándose. Sin embargo, tiene otros inconvenientes como conocer a ciencia cierta el orden de las operaciones, especialmente en parches de gran tamaño. También, cuando el tamaño crece, es difícil llevar una organización clara y entendible. No obstante, la posibilidad de integrar parches más pequeños en otros parches para realizar ciertas operaciones (similarmente al caso de funciones o procedimientos en lenguajes de alto nivel) ayuda a mantener una estructura organizada [31].

### 1.2.2 Unity 3D

Unity es un poderoso entorno de desarrollo con una gran variedad de herramientas que pueden satisfacer las necesidades de un creador de videojuegos u animaciones. El editor es intuitivo y altamente configurable, permitiendo al usuario una gran libertad a la hora de diseñar su espacio de trabajo [32].

Para el interés del proyecto, que es recrear una escena virtual en tres dimensiones y poblarla con efectos de sonido, Unity es una herramienta muy adecuada. Tiene una curva suave de aprendizaje, que permite crear en un periodo de tiempo razonable la mayor parte de los elementos requeridos para la realización. Dispone de múltiples opciones muy útiles para el desarrollo de audio procedural, así como un entorno amigable.

### 1.2.3 Herramientas secundarias

Si bien el grueso del proyecto se basa en la implementación de la escena en Unity 3D y la de los efectos de sonido en PD, otras herramientas secundarias complementarias a estas dos han sido utilizadas a lo largo de la elaboración del proyecto:

-*Matlab:* es un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, la visualización y la programación. Mediante Matlab, es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. El lenguaje, las herramientas y las funciones matemáticas incorporadas permiten explorar diversos enfoques y llegar a una solución antes que con hojas de cálculo o lenguajes de programación tradicionales, como pueden ser C/C++ o Java. Se puede utilizar en una gran variedad de aplicaciones, tales como procesamiento de señales y comunicaciones, procesamiento de imagen y vídeo, sistemas de control, pruebas y medidas, finanzas computacionales y biología computacional. Más de un millón de ingenieros y científicos de la industria y la educación utilizan Matlab, el lenguaje del cálculo técnico [37]. En este proyecto se ha utilizado la punta del iceberg de las posibilidades de Matlab para obtener las representaciones gráficas de las señales temporales y espectrales de los diferentes efectos de sonido.

-*Adobe Audition*: es un completo conjunto de herramientas que incluye funciones de forma de onda, visualización espectral y multipista. Este potente programa de edición de sonido se ha diseñado para acelerar los flujos de trabajo de la producción de audio y vídeo y ofrecer los más elevados estándares de calidad de sonido [38]. Esta herramienta ha sido utilizada por la simplicidad del método de obtención de los espectrogramas en una grabación, útiles a la hora de la implementación de los efectos, para lo que bastaba la opción de prueba.

-*Xamarin Studio*: entorno de desarrollo para diversas plataformas en diferentes lenguajes. En este caso, se ha utilizado para el desarrollo de los “scripts” en C# que dan vida a la escena en Unity 3D.

## 1.3 Motivación

Ya se han comentado las diferentes características de la disciplina del audio procedural, se han expuesto sus puntos débiles y sus puntos fuertes, así como se ha realizado una contextualización sobre en qué situación se encuentra, qué problemas intenta resolver y que direcciones tomará en el futuro.

En un mundo en el que la tecnología informática en general, y las herramientas de desarrollo de videojuegos en particular, avanzan a un ritmo considerable, es necesario proveer de un adecuado sistema de generación de sonido a los productos de entretenimiento electrónico de última generación.

Las técnicas de audio procedural y los resultados que son capaces de generar poseen lo necesario para ocupar un lugar de importancia en estos sistemas, rellenan ciertos vacíos que dejan algunas de las opciones existentes, a la vez que dejan una puerta abierta a la experimentación y la creatividad.

No sólo se trata de mejorar estética y cualitativamente el resultado final, con las técnicas de audio procedural también se puede mejorar el rendimiento del hardware encargado de la reproducción, ya estemos hablando de un ordenador o una videoconsola. Generar audio en tiempo real tiene un coste computacional en ocasiones considerablemente menor (coste variable tratado anteriormente), a la vez que supone un ahorro considerablemente mayor de memoria de almacenamiento. En tiempos en los que el hardware tiene que mover motores gráficos capaces de representar entornos y personajes muy detallados, a la vez que maneja un número cada vez mayor de elementos diferenciados, cualquier ahorro en este sentido será muy bienvenido para los desarrolladores.

Más allá de lo ya mencionado, estamos ante un campo de estudio relativamente joven y poco explotado, con un potencial muy grande, en el que con matemáticas y conocimientos de desarrollo y procesado de señal se pueden realizar avances y aportes que marquen la diferencia.

## 1.4 Objetivos

Habiendo dejado ya claros los motivos que han llevado a la realización de este proyecto, se exponen a continuación los objetivos a satisfacer y que serán posteriormente discutidos.

1. El estudio y la inmersión en el campo del audio procedural, así como de las principales herramientas de desarrollo y posibilidades.

2. Recreación de una cantidad de efectos de sonido utilizando las técnicas de audio procedural mediante dichas herramientas de desarrollo.

3. Integración de dichos efectos de sonido en una representación en tres dimensiones de un entorno natural. Está escena estará protagonizada por un personaje que interactuará con elementos ajenos a él y que, en consecuencia, generará los efectos de sonido ya comentados para intentar provocar una sensación de inmersión e identificación con el usuario.

# Capítulo 2: Estado del arte

## 2.1 Historia y nombres relacionados con el audio procedural

El audio procedural entendido como alternativa para proporcionar efectos de sonido a videojuegos frente a la tradicional grabación en estudios o el audio sintético generado mediante chips y tarjetas de sonido tradicional de las décadas de los 80 y 90 es relativamente joven. No existe demasiado conocimiento de la disciplina y el número de desarrolladores que le dedican tiempo y esfuerzo es bajo a día de hoy. El investigador y programador británico Andy Farnell con su libro "Designing Sound" es la figura más representativa del campo, la que ha profundizado más en la teoría y las técnicas que se emplean para en la materia. Perry Cook, investigador de la Universidad de Princeton, autor del "Sysnthesis Toolkit" y de relevantes libros de síntesis de audio en tiempo real, Kees van den Doel, investigador y experto en parametrización y modelado o Eduardo Reck Miranda, autor del libro "Computer Sound Design" y experto en implementaciones para música procedural y síntesis de audio son otras figuras importantes de la disciplina[1].

## 2.2 Pisadas

Uno de los efectos de sonido más populares a la hora de trabajar y que engloba más formas diferentes de atacar el problema es el de las pisadas. Perry Cook propone una solución basada en el análisis de grabaciones sobre diferentes superficies, extrayendo el tempo y las asimetrías entre las pisadas entre el pie izquierdo y el derecho, y un postprocesado en base a estos resultados [2]. Dicho procesado consiste en extraer la envolvente de control de la grabación original, un modelado de partículas denominado "PhISEM" (Physically Inspired Stochastic Event Modelling) en el que se aprovecha que el origen del sonido está, precisamente, en la interacción entre partículas provocada por la fricción y la presión ejercida por el pie al suelo, y la síntesis final. El objetivo de Cook era establecer una comparativa entre las grabaciones reales y los sonidos sintéticos, remarcando al final que es un buen punto de partida, aunque quedaría trabajo por hacer en lo que se refiere a la extracción de la envolvente, la resonancia y extracción de los parámetros de partículas.

Bresin, Friberg y Dahl proponen en su trabajo un modelo para controlar efectos de sonido basados en modelos físicos [3]. Para ello utilizan grabaciones de sonidos de pisadas caminando y corriendo, sobre los que realizan unos estudios frecuenciales y temporales, además de analizar otras características como el tempo.

Bresin, esta vez junto a Fontana, y apoyándose en su trabajo anterior, implementó en Pure Data un sintetizador de sonidos de pisadas y crujidos de latas aplastándose [4]. La implementación consiste en generar un controlador del sonido en base al tempo de paso y a si el sujeto está caminando o corriendo, y aplicar las características frecuenciales previamente estudiadas.

Pathon, en su tesis para completar el Máster en la Universidad de Limerick, utilizando Pure Data y Python implementó un modelado de pisadas utilizando "síntesis substractiva" que responde dinámicamente a las interacciones entre el usuario y el videojuego y explorar la aplicabilidad del audio procedural en dicho campo[5]. Para ello se basó en el trabajo de Farnell (del que obtuvo personalmente consejo directo), Cook, Fontana y Morreale.

## 2.3 Efectos relacionados con el agua y fluidos

Siendo el agua el fluido por excelencia, se puede aplicar el conocimiento físico existente para generar efectos sonoros producidos por la interacción del líquido con otros elementos bajo diferentes circunstancias. Zheng y James propusieron un método de síntesis automática de sonidos basados en la creación de burbujas, vibración, radiación y advección [6]. El sistema simula un flujo con burbujas, para cada burbuja aproxima la superficie de vibración entre el fluido y el aire y la presión sonora resultante y realiza una superposición lineal para el oyente.

Milavcic, Zita y Arvidsson aplicaron el modelado físico para la síntesis de un sonido de lluvia. Implementaron un simulador en tiempo real de grandes celdas de gotas de lluvia impactando sobre superficies sólidas y líquidas [7]. Aprovecharon la similitud entre el sonido del impacto de una gota y el de la explosión de una burbuja para su sistema, además de modelos matemáticos partiendo de ecuaciones de ondas. Añadieron a la representación un equipo de múltiples altavoces, situando al oyente dentro de un área delimitada por ellos, variando el volumen de cada altavoz a la hora de representar el sonido de una gota, añadiendo percepción espacial.

Van den Doel también basó su trabajo en el sonido producido por la emisión acústica de burbujas. Desarrolló una aplicación que producía sonidos de burbujas únicas como base, para luego implementar un modelo estocástico de síntesis interactivo en tiempo real que simulaba sonidos más complejos como riachuelos, lluvia, ríos u olas rompiendo [8]. En base al sonido básico de una burbuja, van den Doel estudió como combinarlos hasta llegar a los sonidos más complejos descritos anteriormente, mediante la modificación de diversos parámetros en tiempo real. La aplicación ofrece una interfaz en la que se pueden combinar al gusto y comprobar los diferentes resultados.

En un estudio conjunto, Moss, Yeh, Hong, Lin y Manocha realizaron otra aproximación hacia la síntesis de sonidos de fluidos. También partiendo de la base de las burbujas, acoplaron ecuaciones basadas en modelos físicos referidos a la resonancia de las burbujas con múltiples simuladores de fluidos [9]. Como novedad, introdujeron generalizaciones para casos de burbujas de forma no esférica.

## 2.4 Fuego

El fuego es otro fenómeno físico susceptible de ser sintetizado mediante modelado físico. Los japoneses Dobashi, Yamamoto y Nishita propusieron un método para generar el sonido mediante un campo de turbulencias donde el movimiento complejo de vórtices llevaba a la generación de sonido [10]. Previo al proceso, se creaban unas texturas de sonido para los vórtices, que luego eran utilizadas para la representación de los sonidos.

Chadwick y James implementaron un método para sintetizar sonidos de fuego sincronizados con animaciones basadas en modelos físicos [11]. Dividieron su sistema en dos partes: un sonido de llamas a baja frecuencia que recibía datos de una animación reproducida a una relativamente baja frecuencia de muestreo, y dos bandas adicionales de alta frecuencia que encajaban con los datos teóricos obtenidos previamente de grabaciones.

Drettakis y Verron realizaron un sintetizador que representaba sonidos ambientales basados en partículas. Partiendo de lo que llamaban “sonidos atómicos” que pueden ser parametrizados y distribuidos estocásticamente en el espacio y tiempo, representaron modelos para diversos fenómenos ambientales, el fuego entre ellos [12]. Sintetizaron el sonido del fuego como una combinación de impactos ruidosos simulando crujidos, y ruido ecualizado para simular la combustión.

## 2.5 Animales

Los sonidos producidos por animales han sido objeto de investigación por los desarrolladores de audio procedural. Desde el zumbido de las alas de un insecto hasta el gruñido de un mamífero han sido reproducidos mediante estas técnicas. El canto de los pájaros, debido al amplio rango de timbres y frecuencias, es un interesante campo en el que diversos investigadores han hecho aproximaciones. Fagerlund investigó la manera de producir canto de pájaros a través de las técnicas tradicionales de reproducir voz humana [13]. Estudió anatomía de dichos animales, realizó un estudio espectral de diferentes cantos y desarrolló dos modelos físicos para dos tipos de siringe. Kahrs y Avanzini examinaron los mecanismos acústicos de la producción de sonidos en pájaros, los contrastaron con su anatomía y simularon el sonido de un pájaro psitácido [14]. Smyth y Smith III presentaron un modelo del trayecto vocal de las aves utilizando síntesis por guías de onda para la tráquea, métodos numéricos y un modelo con la ecuación de Bernoulli para el desplazamiento de la membrana de la siringe [15]. Estos mismos autores, junto a Abel, participaron en el desarrollo de un modelo para extraer los parámetros de un modelo de síntesis basado en la simulación de la siringe aviar a partir de grabaciones de cantos de pájaros [16]. En este modelo, en cada lapso de tiempo, un ratio normalizado de similitud se introduce en una matriz que representa la relación presión-tensión indicando dicha similitud entre el canto pregrabado y el espectro tabulado. Sucesivas matrices de presión-tensión son almacenadas y los cambios producidos entre ellas son utilizados para controlar el modelo. El modelo acaba simulando la válvula de presión que forman en la tráquea la membrana y el cartílago.

En cuanto a los sonidos producidos por insectos, Brothers y Tschuch modelaron la vibración y sonidos producidos por los aleteos mediante un modelo estridulatorio básico, donde un sonido estridulatorio es aquél que es producido mediante fricción o roce de dos superficies [17]. Para probar los resultados, investigaron los sonidos producidos por una clase de avispa con la ayuda de métodos analíticos. Yendo más allá, de nuevo Smyth y Smith III partieron de la base de que el entendimiento de la producción de sonidos en el mundo animal podría proveernos de modelos para crear nuevos instrumentos musicales para modelar el mecanismo de sonido de la avispa "cicada" y generar una herramienta para producir música mediante un guante MIDI para recrear los diferentes parámetros [18].

Otra rama de la síntesis de sonido que despierta interés entre los investigadores es la recreación de voz en humanos o diferentes tipos de sonidos de otros mamíferos. Cook investigó en su tesis la posible síntesis de canto a través de un modelo de trayecto vocal humano que, mediante filtros digitales, simula tubos acústicos [19]. Por otra parte, Martino creó un sintetizador de llamadas de animales, con posibilidad de recrear sonidos de diferentes especies, tanto mamíferos como reptiles, aves o anfibios, e, incluso, de otros elementos imaginarios asociables a aliens o robots [20].

## 2.6 Objetos sólidos

Los investigadores relacionados con el audio procedural han tenido como objetivo siempre poder recrear los sonidos resultantes de la interacción entre dos objetos, para hacer posible una posterior reproducción realística en ambientes digitales. En esta línea, varios autores han realizado diferentes aproximaciones. Van den Doel, Kry y Pai describieron algoritmos para síntesis en tiempo real de efectos de sonido para simulaciones interactivas como videojuegos y animaciones [21]. Estos efectos eran producidos a partir de modelos 3D automáticamente utilizando simulación dinámica y la interacción de un usuario. Los sonidos consistían en contactos continuos entre dos elementos. Esta vez van den Doel y Pai utilizarón la síntesis modal para recrear objetos vibrantes, donde estos objetos son modelados por un banco de osciladores amortiguados excitados por un estímulo externo [22]. Otra técnica de generación de sonido en tiempo real de movimientos de objetos solidos es la implementada por O'brien, Shen y Gatchalian, precomputando numéricamente la forma, y modos de deformación de un objeto de forma arbitraria, basados en una descripción geométrica y parametrización de materiales y utilizando los datos provistos por una simulación dinámica [23]. James, Barbic y Pai describieron un nuevo algoritmo para la síntesis de radiación de sonido realista de objetos rígidos [24]. Partieron de la precomputación de las vibraciones lineales de cada objeto, relacionando cada modelo a su campo de presión o función de transferencia acústica utilizando métodos estándar de acústica numéricos. Un grupo de investigadores italianos, entre los que se encuentra el ya mencionado Fontana, dieron un paso más allá, implementando modelos de sonidos "cartoonizados" de sonidos producidos por la interacción entre objetos [25]. Bonneel, Drettakis, Tsingos, Viaud-Delmon y James realizaron un modelo de síntesis modal de alta eficiencia computacional basado en cálculos con transformadas "cortas" de Fourier en lugar de ser realizados en el dominio temporal [26]. Zheng y James profundizaron más en sonidos modales producidos por vibración y fricción en su trabajo introduciendo elementos como el sonido producido por elementos no rígidos o el cambio de energía vibracional [27].

# Capítulo 3: Efectos de sonido en Pure Data

## 3.1 Programación gráfica en Pure Data

PD ha sido descrito como entorno de programación gráfica. Esto significa que las líneas de código utilizadas en la programación convencional para elaborar funciones e interactuaciones entre ellas son sustituidas por objetos visuales que pueden ser manipulados en la pantalla. Los usuarios de PD pueden crear nuevos programas (parches) situando funciones (objetos) en la pantalla. Pueden cambiar la forma en la que se comportan enviándoles mensajes y conectándolos de diferentes maneras dibujando cables de conexión entre ellos.

La mayor ventaja que existe en la utilización de PD reside en la posibilidad de modificar parámetros, conexiones, e incluso objetos durante la ejecución en tiempo real, pudiendo así comprobar en el mismo instante el efecto de los cambios introducidos por el usuario, siendo muy útil esta característica a la hora de intentar componer o generar un sonido concreto. A continuación, se ofrecen unas nociones básicas del lenguaje para ayudar a la comprensión del lector de las figuras que contengan parches.

### 3.1.1 Objetos y conexiones

Los objetos son los sustitutos de las líneas de código en la programación gráfica de PD. Generalmente, tienen entradas y salidas, aunque podrían tener exclusivamente una o más de una de ellas. Las conexiones pueden trasmitir variables en sus múltiples formas (enteros, reales, mensajes, flags, “bangs”) o señales. En el primer caso, el cable de conexión será una delgada línea negra, y en el segundo, una línea gris más gruesa. En la Figura 3.1 vemos un ejemplo de conexión e interacción entre objetos.


Figura 3.1: Ejemplo de conexión entre objetos

En este ejemplo, tenemos un objeto principal, [osc~], que representa un oscilador. El número de entrada indica la frecuencia impuesta al oscilador, en este caso, los 440Hz de la nota “La”. El tercer objeto, [dac~], es un conversor digital-analógico que transforma la señal digital recibida en analógica para que sirva como salida de sonido. En este objeto, cada entrada representa un canal. Así, el sonido producido por este parche es una nota “La” continua de 440Hz.

### 3.1.2 Mensajes, símbolos y comentarios

El “mensaje” es utilizado para almacenar y mandar información a otros objetos, y puede contener números o texto. Tiene una forma particular, similar a la de un sobre de cartas. También es posible enviar números y otra información al mensaje.

Un símbolo es otra manera de almacenar y enviar información. Una vez creado, es posible usarlo para mostrar la salida de otros objetos, o escribir directamente en él y pulsar “enter” para enviar el texto. El espacio es considerado como una separación entre símbolos.

Un comentario es, al igual que en la programación convencional, una manera de tomar notas sobre el código para hacerlo más entendible para uno mismo o una tercera persona. Tienen como peculiaridad que no están encerrados en ninguna caja. No tendrán efecto sobre el parche. En la Figura 3.2 vemos un ejemplo de cada elemento.

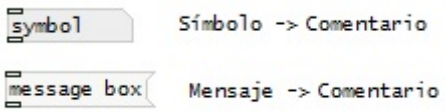


Figura 3.2: Mensaje, símbolo y comentario

### 3.1.3 Objetos GUI

A continuación, se detallan los principales objetos GUI utilizados en la implementación de los parches de PD.

*Número*: cuando en un símbolo tenemos un dato numérico, tenemos lo que se conoce como “número”. Permite mostrar un número o introducirlo por medio de teclado o ratón (haciendo clic sobre el objeto, y arrastrando hacia arriba o hacia abajo para aumentarlo o disminuirlo). Sus límites pueden ser editados en sus propiedades.

*Bang*: este objeto envía un mensaje llamado “bang” cada vez que se hace clic en él. “Bang” es un mensaje especial, el cual muchos objetos interpretan como “haz una acción ahora mismo”. Utilizar este objeto GUI equivale a utilizar un mensaje con la palabra “bang” dentro. También puede ser utilizado para recibir y enviar mensajes “bang”.

*Toggle*: cuando es clicado, envía un “cero” si su casilla no está marcada, y un “número no igual a cero” (por defecto uno, aunque puede configurarse de otra manera) si su casilla lo está. También posee una entrada que permite determinar si un número de entrada es o no “cero”.

*Vslider y Hslider*: son “sliders” verticales y horizontales que envían su valor actual cuando son movidos con el ratón. Su rango por defecto es de 0-127, puesto que está pensado para la tecnología MIDI, pero puede ser cambiado en sus propiedades. Ambos tienen una entrada que puede ser utilizada para mostrar valores entrantes dentro del rango del slider. En la Figura 3.3 se muestran los objetos GUI aquí mencionados.

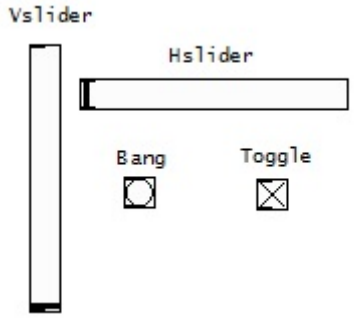


Figura 3.3: Objetos GUI

Las propiedades visuales de estos objetos pueden ser modificadas para hacer los parches más amigables y fácilmente entendibles, y también añadirles etiquetas.

### 3.1.4 Principales objetos utilizados

[noise~]: Objeto que sirve para generar ruido blanco de distribución uniforme. Aunque posee una entrada, está inactiva y no tiene ninguna utilidad. Su rango de salida de valores va de -1 a 1 y será utilizado como fuente de ruido a dar forma en los diferentes parches.

[dac~]: Conversor digital-analógico, recibe una señal digital que transforma en analógica para la salida principal de audio. Cada una de las entradas corresponde a un canal, aunque en la totalidad de los parches diseñados a lo largo del proyecto, por ambas se manda la misma señal.

[print]: Este objeto imprime los mensajes que le llegan por la entada en la ventana del terminal de PD. Es posible añadirle texto junto a "print" que también imprimirá en cada mensaje.

[inlet]/[inlet~] y [outlet]/[outlet~]: Estos objetos definen entradas y salidas para parches encapsulados en otros parches. Los que van acompañados de "~" sirven para entradas y salidas de señales y los otros para variables corrientes. Si definimos un subparche dentro de un parche nombrando un objeto cualquiera con "pd"+ texto y creamos objetos de este tipo dentro de él, aparecerán en el objeto del parche superior tantas entradas o salidas como "inlets" o "outlets" hayamos definido.

[switch~]: Este objeto tiene como único elemento de interacción una entrada, por la cual, si recibe algo diferente a 0, hará que el parche en el que esté incluido se ponga a funcionar. En este estado, si recibe un 0, detendrá el funcionamiento del parche. Útil para controlar un sonido que queremos que esté presente eventualmente.

[*~]/[/~]/[+~]/[-~]: Estos objetos son operadores de señales que las combinan con escalares. Se pueden añadir números en la misma caja del objeto, o configurarlos como parámetros de entrada. Tienen dos entradas, la de la señal a operar, y un posible operador numérico, y una salida, la de la señal una vez realizada la operación.

[lop~]/[hip~]/[bp~]/[vcf~]: Son filtros paso-bajo, paso-alto, paso-banda (de un polo) y controlado por tensión respectivamente. Sus parámetros pueden indicarse en la caja del objeto, pero también por las entradas, siendo una opción más interesante debido a que en ocasiones se requerirá que las bandas y las frecuencias de corte cambien. Cabe mencionar que, por un bug en la programación del objeto [vcf~], no es posible introducir sus valores iniciales escribiéndoselos en la caja. Es necesario utilizar mensajes o números. Por ello, a lo largo de la implementación, es recomendable ignorar los valores que aparecen en la caja de dicho objeto.

[clip~]: limita una señal entrante entre dos valores. Es útil si queremos que un señal no sobrepase un cierto umbral o si queremos que una variable no sea mayor o menor que un cierto valor límite.

### 3.1.5 Grabaciones

A lo largo del desarrollo del proyecto, se han utilizado grabaciones digitales descargadas de Internet de los distintos fenómenos naturales recreados para su estudio y posterior implementación. Dado el carácter académico de este trabajo, se ha decidido no hacer ningún tipo de inversión económica en la adquisición de ninguno de estos efectos, salvo que se hubiera llegado al caso de excesiva necesidad. Como medida en primera instancia, se decidió en su momento grabar estos sonidos “in situ”, tomando como medida provisional la expuesta, y quedando como definitiva tras llegar a la conclusión de que estas grabaciones tenían la calidad suficiente para seguir adelante con la síntesis de los efectos.

Todas las grabaciones fueron descargadas a través de las páginas web “www.freesfx.co.uk”, “www.freesound.org” y “www.youtube.com”.

## 3.2 Cascada

Para elaborar un sonido que imitase la caída natural del agua en una cascada se ha procedido primero al estudio frecuencial de una grabación digital de dicho fenómeno. En las figuras que siguen, tenemos su representación en los dominios del tiempo y de Fourier.

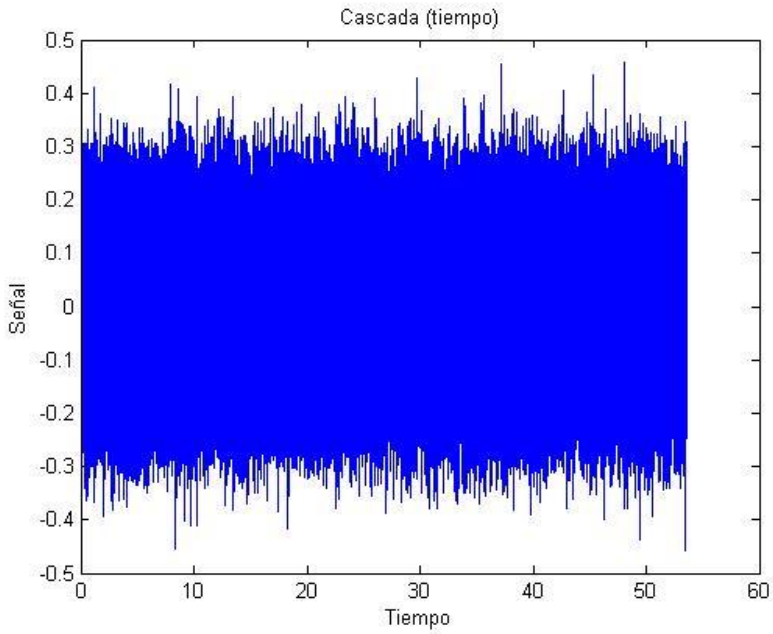


Figura 3.4: Sonido de cascada en el tiempo

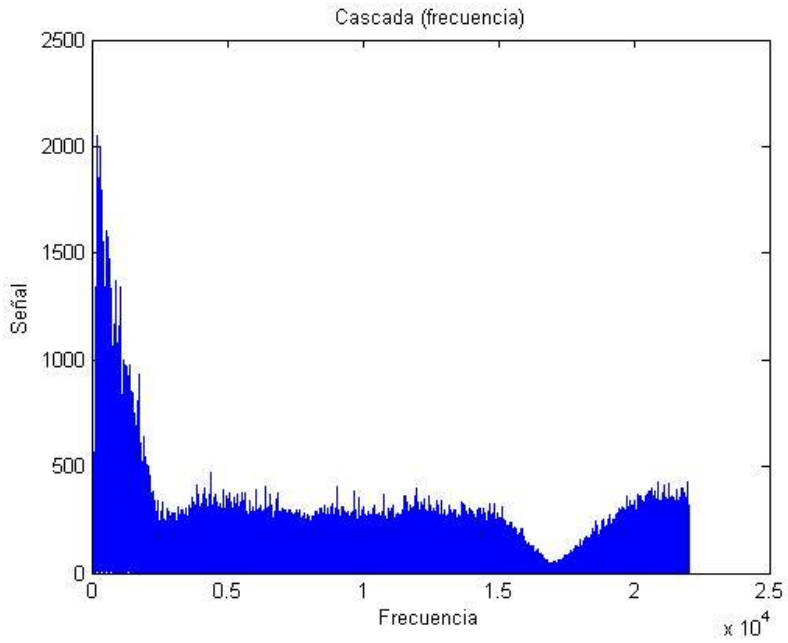


Figura 3.5: Sonido de cascada en frecuencia

De su señal en tiempo no se pueden extraer muchas conclusiones, solo que es un sonido de muchas variaciones, en el que hay presentes una gran cantidad de frecuencias. Sin embargo, sí que se sacan datos interesantes de su representación en frecuencia. Vemos un pico muy pronunciado alrededor de los 200Hz que va descendiendo paulatinamente hasta alcanzar un nivel prácticamente constante a partir de los 2500Hz. A continuación, en la Figura 3.6, podemos ver la implementación del parche en PD.

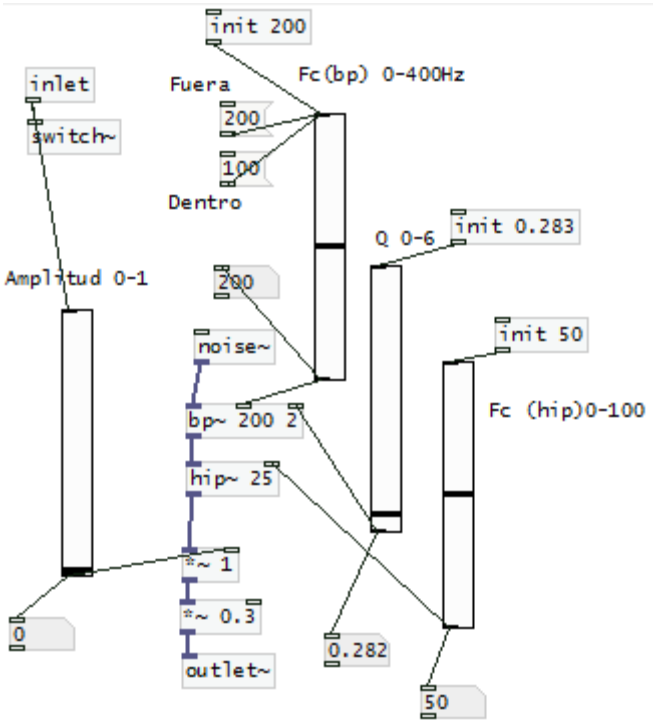


Figura 3.6: Cascada

La representación consiste en un filtrado paso-banda, centrado en 200HZ y con un valor del factor de calidad del filtro bajo (0.283) que permite un corte suave, y un posterior filtrado paso-alto con frecuencia de corte de 50Hz que evita la aparición de las frecuencias más bajas de un ruido blanco que ejerce como fuente del sonido. Se han incluido los “sliders” que se utilizaron en la fase de desarrollo para calibrar correctamente el filtrado en relación al sonido final aprovechando la ventaja de PD de poder modificar parámetros dinámicamente durante la ejecución. Durante la misma se observó que, centrado el filtrado en 100Hz, se obtenía un sonido más sordo, muy similar al que se escucha cuando el individuo está en una cueva bajo la cascada, aunque por razones técnicas en la implementación de la escena no se realizó. Esto último es una máxima en la programación con PD. Es moderadamente habitual, cuando se está intentado calibrar los parámetros en un parche, descubrir durante las pruebas sonidos relacionados con el deseado con alguna particularidad. En la aclaración sobre PD se afirmaba que una de las ventajas del entorno era poder modificar los parámetros en la ejecución, y esto es consecuencia de ello.

## 3.3 Riachuelo

El sonido que se pretendía conseguir quería imitar al de los pequeños ríos que suelen correr en los paisajes montañosos, donde en su recorrido se encuentran una gran cantidad de rocas y de pequeños saltos de agua, que produce un efecto de fluidez. Al igual que en el resto, el primer paso fue analizar una grabación que fue considerada lo suficientemente adecuada a lo que se quería conseguir.

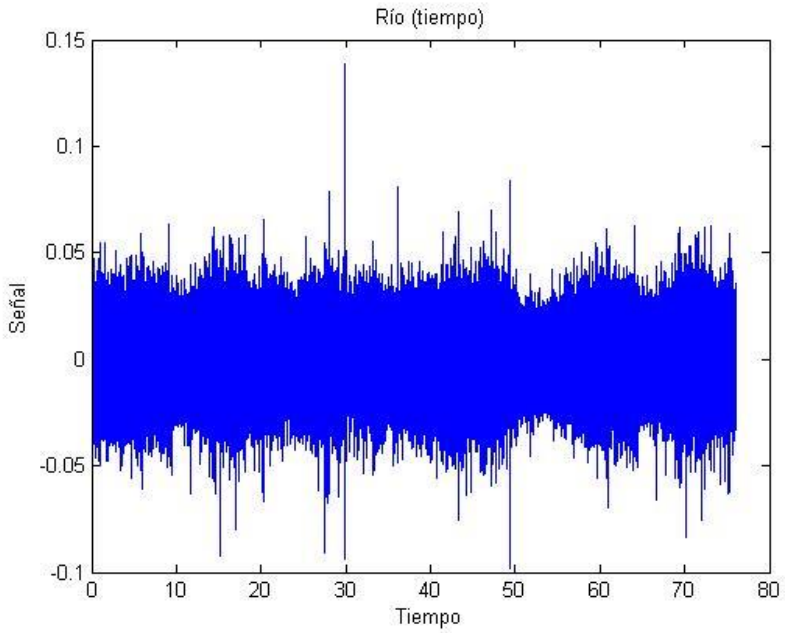


Figura 3.7: Sonido del riachuelo en el tiempo

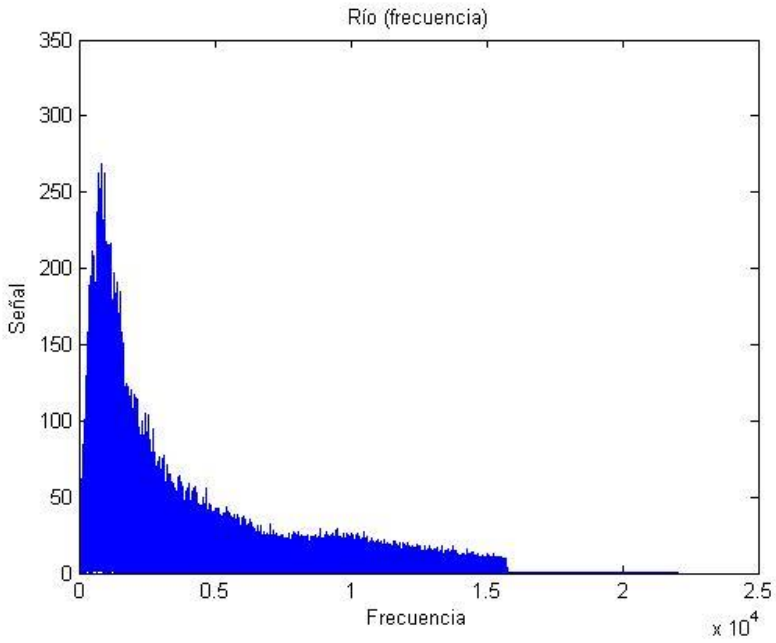


Figura 3.8: Sonido del riachuelo en frecuencia

En la grabación se observó, en primera instancia, un ruido de fondo producido por los millones de burbujas que se generan al colisionar el agua con las rocas, orilla y demás elementos de interacción. Mientras que observamos grandes variaciones a alta y baja frecuencia en la señal temporal, en la señal frecuencial se percibe un pico muy marcado alrededor de 1KHz. Puesto que se lleva la mayor parte de la energía, consideramos que las muestras alrededor de 1KHz son las responsables del sonido de fondo. Ésta es la base de la implementación del parche, expuesto en la Figura 3.9.

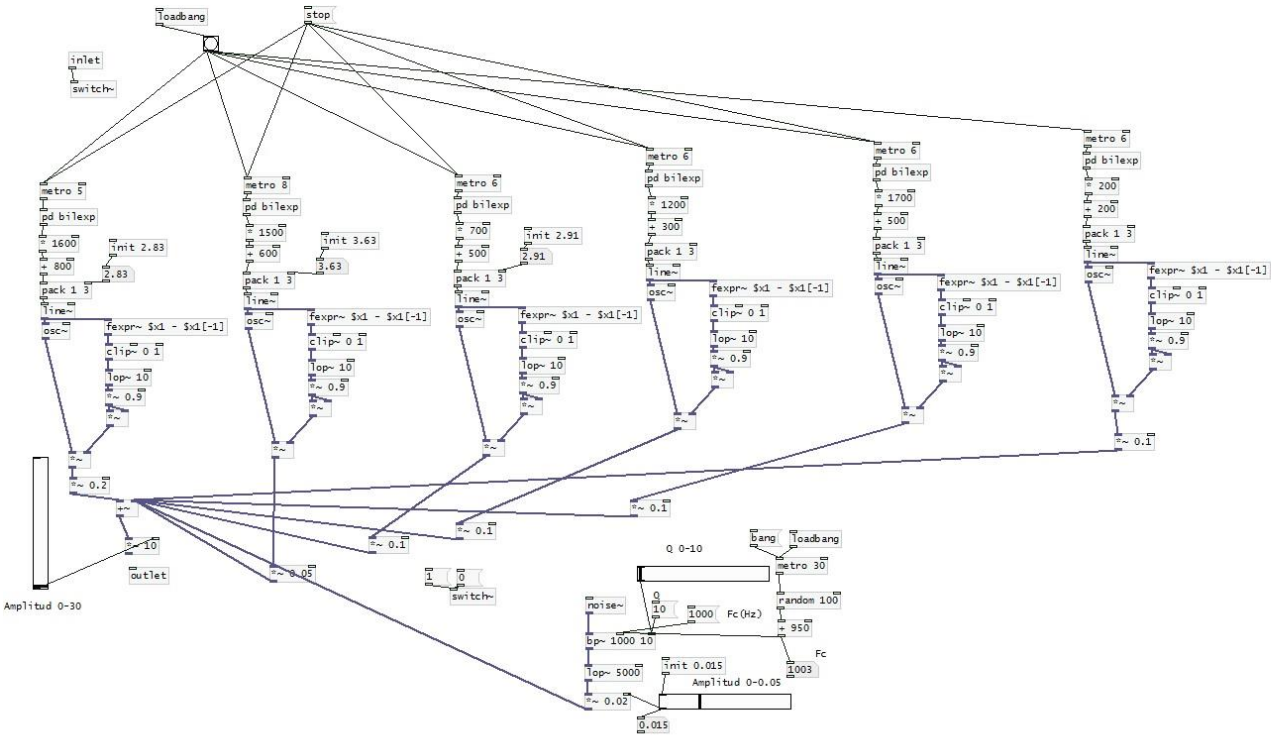


Figura 3.9: Río

En el parche se observan dos partes bien diferenciadas. Por un lado, la inferior, utilizada para producir el ruido de fondo. Por otro lado, tenemos seis columnas de similar aspecto que producen el efecto de fluidez.

En la de generación del ruido, se incluyó un generador de números aleatorios entre 950 y 1050 que determina la frecuencia central del filtro. De esta manera, se aporta riqueza y aleatoriedad al sonido. El filtro es un paso-banda, centrado en 1KHz como se ha especificado en el estudio frecuencial, que toma como entrada un ruido blanco. Posteriormente, se le aplicó un filtrado paso bajo para atenuar más altas frecuencias que quitaban algo de realismo al resultado final.

En la Figura 3.10 observamos una de las columnas del resto de la implementación.

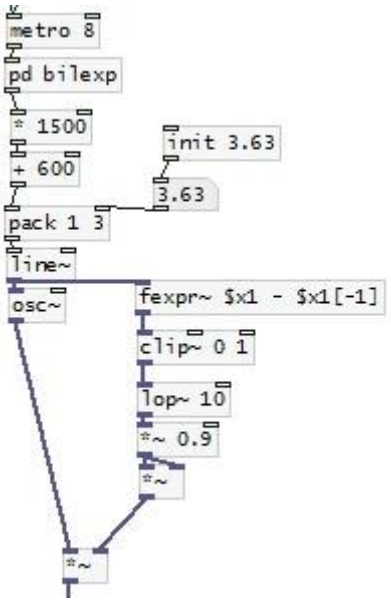


Figura 3.10: Columna del parche del río

El objeto [metro 8] es utilizado para lanzar aleatorios en un intervalo de ocho segundos, no obstante, estos números no son completamente aleatorios. El objeto [pd bilexp], cuya composición puede observarse en la Figura 3.11, contribuye a la implementación dotando a los números de una distribución bilineal exponencial, dada la continuidad del sonido y suponiendo que una frecuencia que aparezca inmediatamente después de otra tendrá un valor cercano al anterior y no excesivamente lejano. De esta manera, a la salida tendremos números positivos y negativos con una alta tendencia a estar cerca de cero [33]. Mediante el objeto [pack] y sus argumentos, limitamos el tiempo que debe tomarse un número para cambiar entre dos valores, lo que favorece la sensación de fluidez, y mediante el objeto [line~] se crea la señal que controla la frecuencia del oscilador.

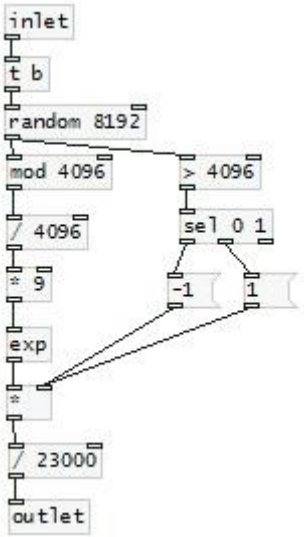


Figura 3.11: pd bilexp

Posteriormente, el oscilador es modulado por una señal diferencial, que toma las muestras actual y anterior de la propia señal que controla la frecuencia del oscilador y es

filtrada dejando pasar las bajas frecuencias, ralentizando de esta manera las variaciones en la señal resultante.

La razón por la que tenemos cinco columnas similares a ésta, en las cuales se cambian ligeramente los valores, es para aportar riqueza al sonido final. Mientras una de estas columnas suena como un único y pequeño flujo, el sonido generado por las seis se asemeja a un flujo mayor en el cual existen pequeños flujos originados por los elementos que se interponen entre el agua y su camino.

## 3.4 Viento

El sonido que produce el viento es muy particular, puesto que se produce por la interacción entre el gas moviéndose entre dos puntos con diferente presión y los objetos que se va encontrando por el camino. Puesto que la escena que estamos generando tiene una carácter marcadamente tormentoso, se ha dispuesto que el viento que tenga una presencia notable. El efecto cuenta con dos modos de funcionamiento, en uno se genera una brisa suave y en el otro se simula un fuerte vendaval propio de una tormenta. En la Figura 3.12 se puede observar la implementación final.

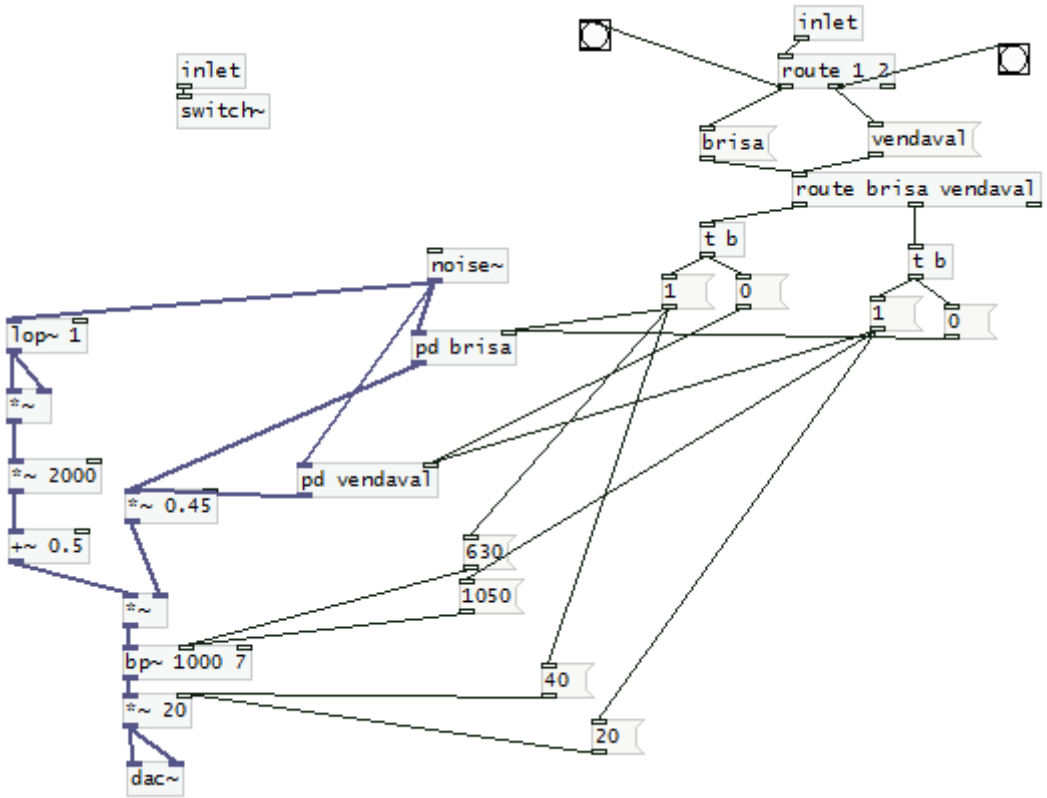


Figura 3.12: Viento

Para sintetizar propiamente un sonido de viento, se debería implementar un filtro por cada objeto con el que interactuara el gas en su desplazamiento. No obstante, esto sería muy costoso y complicado en el caso de contar con ambientes con gran cantidad de elementos. Cuando el aire se está moviendo, lo hace en una única dirección, en ambos sentidos a un ritmo

bajo en comparación con las ondas de sonido. El sonido producido entonces por el desplazamiento es de una frecuencia muy baja, del orden de mHz, fuera de nuestro rango de audición. Sin embargo, cuando interacciona con objetos, se producen turbulencias, que es lo que provoca su sonido característico [33]. Estas turbulencias producen un aullido característico que, cuando sopla moderadamente, está concentrado en frecuencias más bajas y con variaciones más lentas que cuando lo hace con más violencia. Como se explica más adelante, durante la ejecución de la escena se puede escoger entre las dos posibles velocidades del viento, el parche de la Figura 3.12 se encarga de activar la opción escogida. En la Figura 3.13 se muestra el subparche [pd brisa].

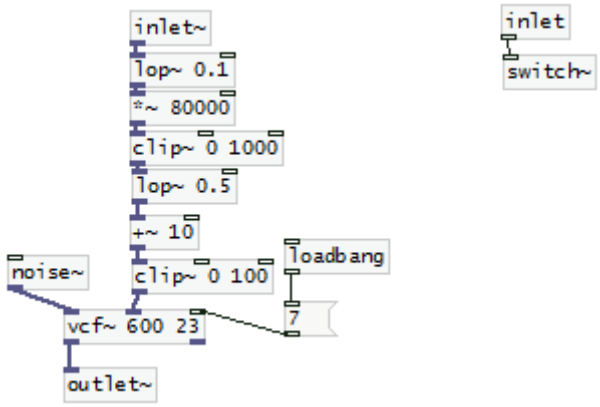


Figura 3.13: Brisa

El parche consiste en una señal ruidosa filtrada paso-banda en un rango frecuencial comprendido aproximadamente entre 500 y 800Hz. Con una señal moduladora filtrada a un nivel tan bajo, las variaciones son lentas y suaves con pequeños aumentos esporádicos que simulan pequeñas rachas de viento. En la Figura 3.14 se puede observar [pd vendaval].

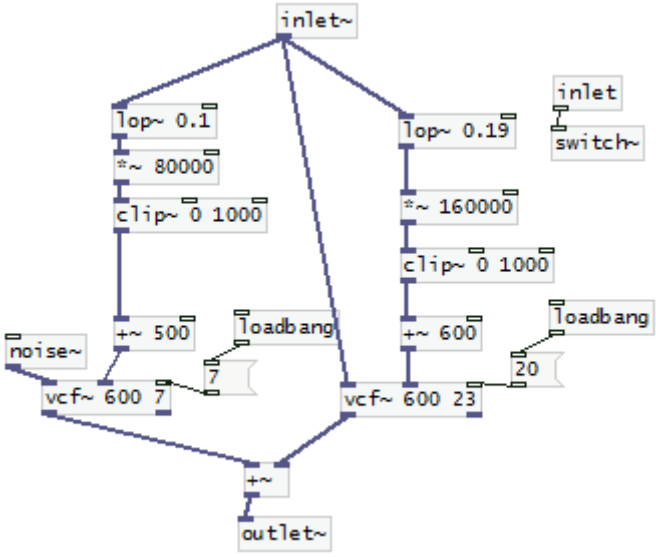


Figura 3.14: Vendaval

Cuenta con una columna muy similar a la de [pd brisa], aunque en este caso el rango está en unos niveles más altos. En la columna de la derecha el funcionamiento es similar, cubriendo las frecuencias de 700 a 1300Hz aproximadamente. Este sonido es más rico y completo, puesto que cubre tanto rachas violentas esporádicas, dando forma a las características turbulencias antes comentadas, como una velocidad más constante y contundente.

Vemos que en el parche de la Figura 3.12 se genera una señal con baja variación (columna de la izquierda) que, multiplicada por la señal resultante de la elección entre la brisa y el vendaval, produce un efecto ondulatorio que añade riqueza al sonido. Evita que, por las variaciones introducidas por las altas frecuencias, la amplitud resultante sea demasiado plana.

## 3.5 Lluvia

El sonido de la lluvia es producido por la colisión de las gotas de agua con los diferentes objetos presentes y el suelo. Estas gotas tienen un diámetro que varía entre 1 y 3 milímetros (mm). Caen con velocidad constante a un ratio de 200 impactos por segundo en un metro cuadrado. En diferentes condiciones, el tamaño, velocidad y ratio pueden variar. Bajo una gravedad y densidad del aire normales, las gotas de 1mm alcanzan alrededor de los 2 metros por segundo (m/s), mientras que las de 5mm pueden alcanzar 10m/s. Sus formas van desde una esfera casi perfecta para las más pequeñas, hasta estar aplastadas verticalmente las más grandes, de forma similar a una hamburguesa [33].

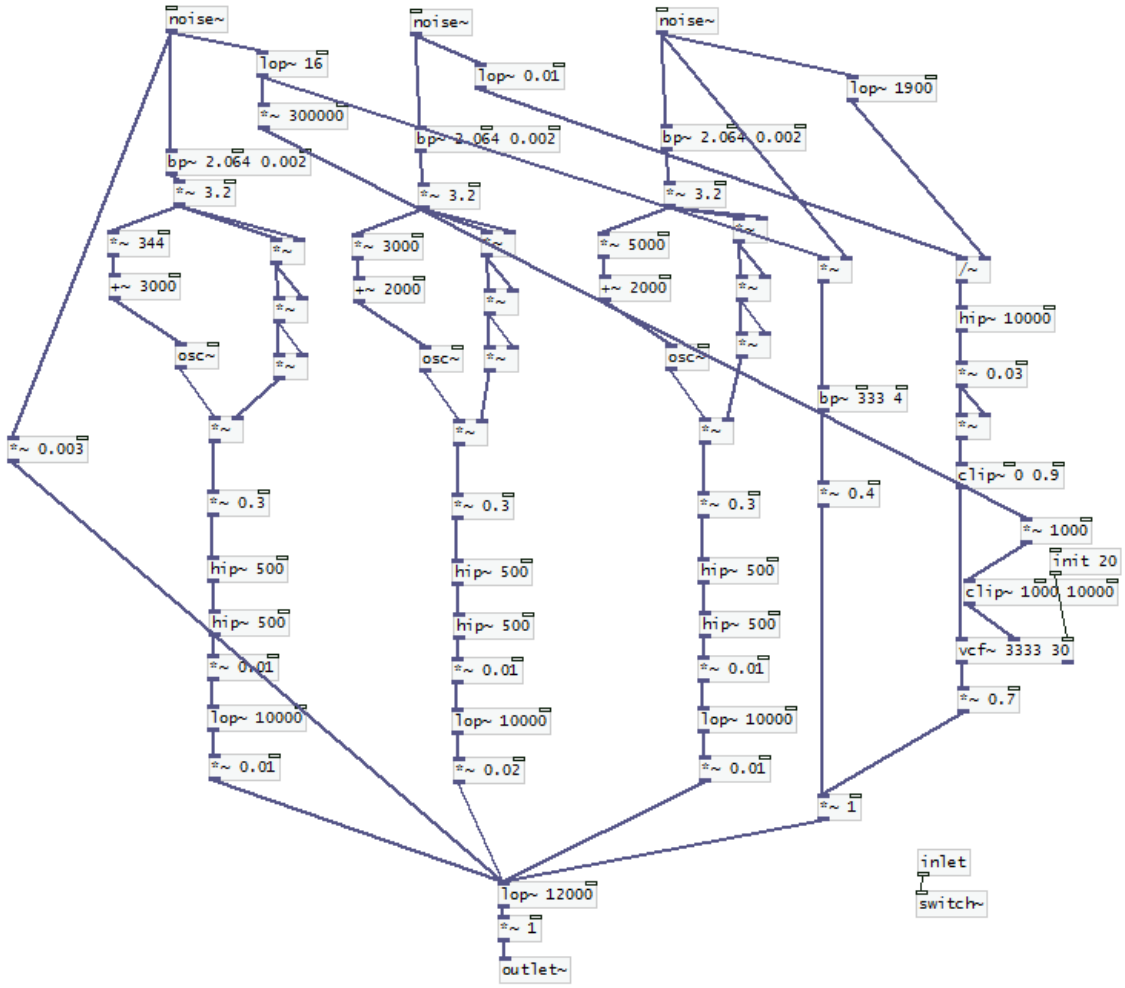


Figura 3.15: Lluvia

El parche está dividido en tres partes, así como son tres tipos de sonido los que se han querido recrear. Por un lado, están las tres columnas centrales, que simulan el goteo principal. Luego está la columna a la derecha de las tres anteriores, que se encarga de reproducir un sonido de baja frecuencia que complementa al resto del parche, típico de cuando el agua impacta con un objeto sólido rocoso macizo. Finalmente, a la izquierda tenemos la parte del parche que proporciona un sonido de fondo. Ésta es, precisamente, la parte más sencilla del mismo, puesto que consiste únicamente en tomar una de las fuentes de ruido blanco y reducir su amplitud hasta que tenga una apariencia testimonial en el resultado final. De esta manera, se consigue un efecto de lluvia lejana, complementando el goteo próximo generado por el resto del parche. Esto es así porque, en un ambiente lluvioso, más allá de las gotas que caen cerca del individuo que está escuchando, hay millones cayendo a una distancia más lejana, que generan sonidos puntuales al chocar con todo tipo de objetos de un amplio espectro de frecuencias.

En cuanto a las tres columnas centrales que simulan el goteo principal, tomando como referencia la de más a la izquierda, se observa que, en el primer paso, si aplica a la señal ruidosa un filtrado paso-banda muy cercano a cero, con un valor del factor de calidad extremadamente bajo. Esto desemboca en una señal en frecuencia con una pendiente descendiente muy suave, escalada posteriormente para que la señal temporal tenga unos

valores cercanos a 1. Posteriormente, es elevada a la octava potencia, lo que provoca un rango dinámico muy alto, con valores muy cercanos a 0 de órdenes muy bajos y valores del orden de la unidad. Esta señal se utiliza para modular la que proporciona un oscilador, cuyos valores de frecuencia central oscilan entre 3 y 10KHz [33]. Utilizando dos veces el mismo filtro paso-alto, se consigue un filtrado con una pendiente muy profunda, con el que se elimina la aparición de componentes indeseadas de altas frecuencias. El resultado es una señal con "clics" de muy corta duración, de diferentes amplitudes y diferentes frecuencias centrales dentro del rango de interés. Las otras dos columnas semejantes a ésta que hemos analizado aportan una señal similar con otros rangos de frecuencias diferenciados, aunque próximos, que sirven para aportar riqueza al sonido al ejecutarse simultáneamente.

En la zona más a la derecha del parche se simulan los goteos de frecuencia más baja y más profundos, que aparecen a un ritmo más bajo que los anteriores. Existen dos columnas diferenciadas. La de la izquierda consiste en una modulación de una señal ruidosa con otra paso-bajo, y filtrada paso-banda posteriormente. Tras la modulación, se obtiene una resonancia ondulante que, tras filtrarla paso-banda con una frecuencia central alrededor de 300Hz, resulta un ruido retumbante [33], similar al que provocan las gotas más gruesas al precipitarse sobre una superficie de objetos voluminosos. Puesto que la escena se representa en un paisaje forestal, donde predomina la vegetación, este sonido está muy atenuado debido a la ausencia de grandes objetos con los que interactuar. Un ejemplo de donde podría tener más relevancia es en un paisaje urbano, donde es habitual escucharlo cuando las gotas caen sobre los coches.

En la última subsección, la columna de más a la derecha del parche, cubre el espectro dejado entre las dos anteriores. Primero, se toman dos señales paso-bajo de dos de las fuentes ruidosas, una con frecuencia de corte en 16Hz y otra en 1900Hz y se dividen muestra a muestra. Esto provoca una presencia en la señal resultante de valores altos de pico, puesto que cuando la señal filtrada en 1900Hz tiende a cero, el resultado de la división tiende a infinito [33]. Esta señal se atenúa, se eleva al cuadrado para evitar valores negativos y se recorta entre 0 y 9, teniendo de esta manera una señal de alta variación entre este rango de valores. Posteriormente, esta señal va a parar a un filtro paso banda, cuya frecuencia central es modulada por otra señal de variación más lenta (ruido filtrado paso-bajo en 16Hz con valores de salida entre 1000 y 10000) que va de 1KHz a 10KHz. El resultado son gotas que resuenan a mayor frecuencia y que caen a un ritmo menor que las anteriores, sin llegar a los niveles de las tres columnas del principio. Dependiendo del ambiente, se puede jugar con los parámetros de cada subsección, especialmente con los de amplitud, para enfatizar un sonido u otro.

## 3.6 Fuego

El fuego es un fenómeno complejo. Es un ejemplo de sonido componible mediante la adición de diferentes partes contribuyentes. El fuego es el resultado de una reacción oxidante fuera de control. Empieza cuando algo que lo alimenta (combustible, madera) se calienta y comienza a oxidarse. Siendo una reacción exotérmica, genera calor. Cuanto más caliente está

el objeto en combustión, mejor se oxida y, cuanto mejor se oxida, más caliente llega a estar. Esta relación se alarga mientras el objeto no se haya oxidado completamente o se corte el suministro de oxígeno necesario para la combustión. Los diferentes procesos en los que se ve envuelto algo que está ardiendo, producen una serie de sonidos:

- “lapping”: combustión de los gases en el aire (llamas).

- “crackling”: pequeñas explosiones producidas por el objeto en combustión.

- “hissing”: producido por los gases liberados

- “bubbling”: producido cuando un líquido hierve

- “creaking”: expansión interna de un gas en un objeto en combustión

- “fizzing”: conflagración aérea de pequeñas partículas

- “whining”: relajaciones periódicas durante la liberación de los gases

- “roaring”: ciclos turbulentos de llamas a baja frecuencia

- “popping”: explosión gaseosa

- “clattering”: asentamiento del gas bajo la gravedad

En términos acústicos, los sonidos predominantes son “lapping”, “crackling” y “hissing”, que han sido recreados por separado [33].

En la Figura 3.16 tenemos la implementación final de “hissing”. El parche es muy básico, puesto que el sonido es puntual y volátil. Se dan violentas variaciones únicamente en pequeños fogonazos, no muy frecuentemente.

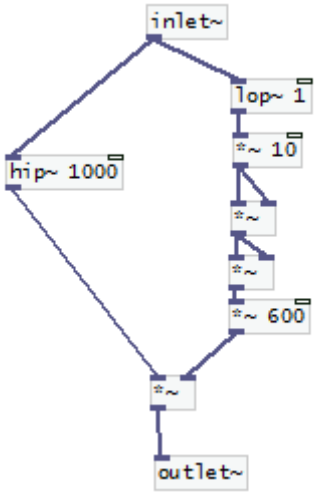


Figura 3.16: Hissing

El primer paso es generar un ruido de fondo en el que las frecuencias bajas estén atenuadas, que es introducido al parche mediante una fuente común a los tres sonidos implementados en el nivel superior, por ello está presente el filtro paso-alto. Posteriormente,

se modula con la señal generada por la columna de la derecha. Esta señal consiste en una señal de baja frecuencia. Al elevarla a la cuarta potencia multiplicándola por sí misma varias veces, se aumenta su rango dinámico, teniendo valores muy cercanos a 0 con órdenes bajos frente a valores cercanos a 1. Tras multiplicar por 600, valor obtenido empíricamente, y multiplicar por la señal con altas frecuencias, obtenemos los “fogonazos” característicos de este tipo de sonido.

El “crackling” es un sonido consistente en crujidos cortos, pequeñas explosiones de madera, carbón u otros sólidos donde un trozo de material se desintegra bajo la presión [33]. En la Figura 3.17 podemos observar la implementación en PD.

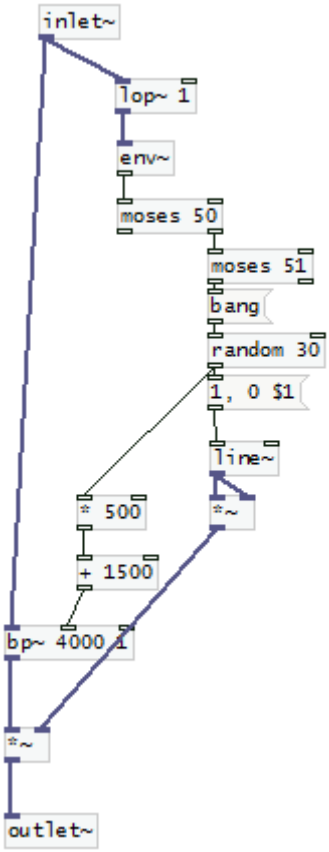


Figura 3.17: Crackling

La idea es generar un parche en el que, esporádicamente, se produzcan unos “clics” que provoquen estos crujidos característicos. Estos crujidos deben estar diferenciados, aunque su frecuencia sea generalmente alta, deben tener una cierta variación en este sentido, así como diferentes duraciones dentro de ser prácticamente instantáneos. Esto se consigue mediante la columna que desemboca en una frecuencia para el filtro paso-banda y en una señal a multiplicar por el ruido filtrado. Mediante una señal de muy baja frecuencia y el objeto [env~] (seguidor de envolvente, proporciona la amplitud RMS de la señal de entrada) se generan números que son filtrados de tal manera que solo pasan a la siguiente etapa los que están entre 50 y 51 gracias a la combinación de los objetos [moses]. Estos números tienen la única función de activar el “bang” que les sigue a un ratio que se aproxima al ratio de un sónido de “crackling” real.  Con el objeto [random 30] se crea un número aleatorio entre 0 y 30

que, por un lado, es escalado para fijar la frecuencia del filtro paso-banda en un lugar en el orden de los KHz. Mientras tanto, con el objeto [line~], se genera una línea de tensión que desciende desde 1 hasta 0 con una pendiente marcada por el número entrante, que marca el decaimiento del crujido al multiplicarlo por la señal, consiguiendo así variedad en esta característica.

Por último, el tercer efecto implementado es el “lapping”. Este efecto es producido por la generación de llamas cuando arde un gas [33]. De los tres, es el parche más sencillo, como podemos ver en su implementación de la Figura 3.18.

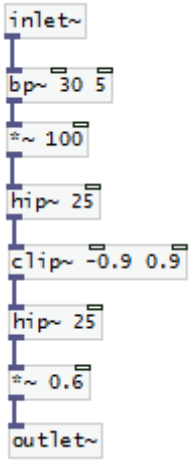


Figura 3.18: Lapping

Al igual que los otros dos efectos, la fuente de sonido inicial para el “lapping” es un generador de ruido blanco. El sonido está formado, principalmente, por componentes de baja frecuencia, de hecho, con un simple filtrado paso bajo a unos 30Hz es posible hacer una aproximación razonable. Sin embargo, aunque pequeña, la aportación de las altas frecuencias es considerable. Por ello, en la implementación definitiva se opta por filtrar el ruido paso-banda con un valor del factor de calidad relativamente bajo. Esta señal es amplificada y filtrada paso alto para eliminar componentes demasiado bajas en frecuencia que pueden contaminar el resultado final. Limitando la señal entre +0.9 y -0.9 se evita que los niveles vayan ocasionalmente más alto de lo que deberían, aunque es algo que no aparecía frecuentemente. El siguiente filtro paso-alto está colocado para evitar el poco probable caso de que la señal se vaya a más de 0.9 por varios valores consecutivos y se bloquee en un valor continuo.

Una vez implementados los tres efectos, se escalan propiamente y se suman. Esto es realizado en el parche “firegenerator”, que se puede ver en la Figura 3.19.

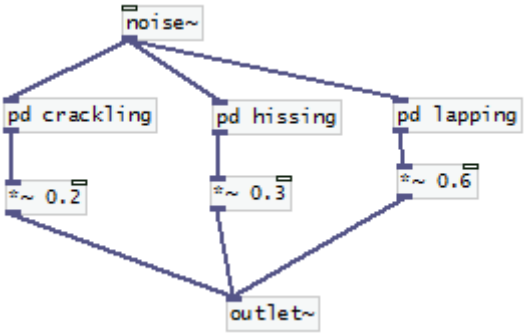


Figura 3.19: Firegenerator

Finalmente, se ha realizado un parche superior, en el cual se combinan varios "firegenerator" para conseguir un efecto más rico. Se puede observar en la Figura 3.20.

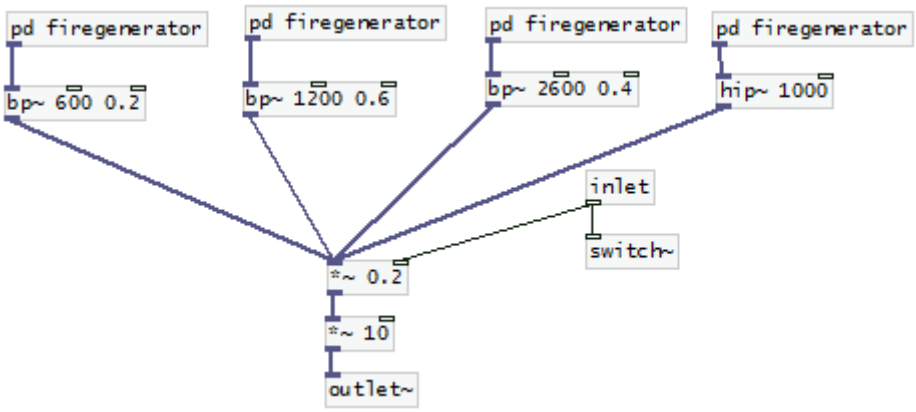


Figura 3.20: Fuego

## 3.7 Truenos

Cuando diez mil millones de Julios de energía son liberados como electricidad, el aire en el camino de la chispa se transforma en plasma, calentado a 30000 grados. Esto provoca que el aire se expanda muy rápidamente, y, dado que la velocidad de la propagación eléctrica está cercana a la velocidad de la luz, sucede simultáneamente siguiendo la senda del rayo, resultando en una radiación cilíndrica hacia afuera. La duración de un rayo es muy corta, así que el aire expandido se enfría rápidamente y se colapsa hacia adentro [33]. El sonido se divide en tres partes: el crujido que se produce en el momento del rayo, el sonido más profundo y grave de las partículas del aire vibrando y las posteriores reflexiones en árboles, edificios, etc.

En la Figura 3.21 se observa el parche que genera el crujido, conocido como "bolts".

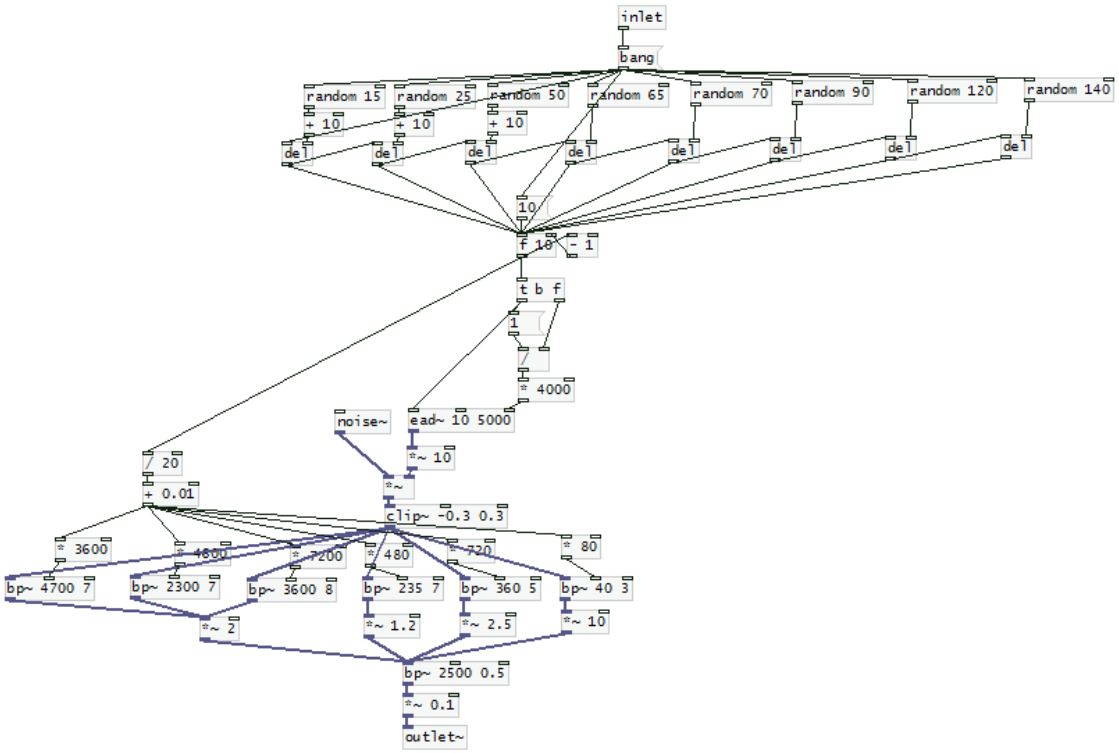


Figura 3.21: Bolts

El parche iniciará su funcionamiento tras un evento que estimule el “bang” de la parte superior. En la parte superior del parche está implementada una forma de propagar mensajes de “bang” con retardos aleatorios, aunque mayores conforme pasa el tiempo tras el evento, como puede observarse al ser los números aleatorios generados con los objetos [random] cada vez mayores. La mayor parte de la energía del crujido es consumida en el primer instante, y va decreciendo durante el corto espacio de tiempo que dura, partiendo de que el contenido en frecuencia está relacionado con la energía, se asume que la “cola” del crujido contendrá componentes más bajas que al inicio [33]. Tras la cadena de retardos, hay implementada una cuenta atrás desde diez, valor que se utiliza tanto para calcular las frecuencias centrales de los filtros del ruido (irán decreciendo conforme avanza el tiempo, cumpliendo la norma antes nombrada) como el decaimiento (objeto [ead~], exponential attack-decay, genera una envolvente que va decreciendo exponencialmente).

El sonido que llega tras la señal directa es generado por la reverberación del sonido original con los elementos del entorno.  Los diferentes rebotes se combinan en fase y se distorsionan para producir una especie de zumbido que varía ondulatoriamente en volumen y tono. Se trata de sonidos graves de baja frecuencia, la implementación final se encuentra en la Figura 3.22, nombrada como “afterimage”.

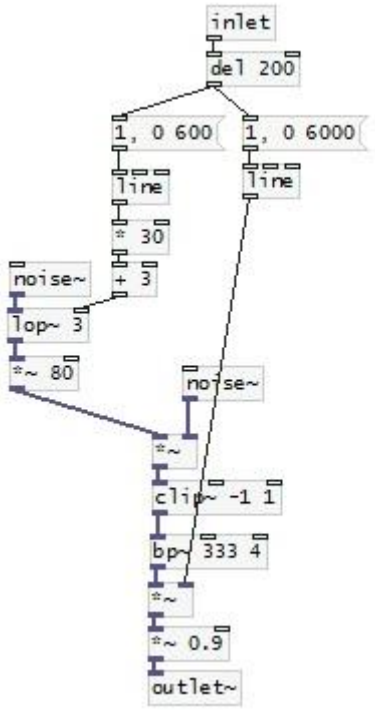


Figura 3.22: Afterimage

El parche tiene un retardo inicial de 200ms para dar tiempo a “bolts” de producir su sonido. En la ramificación de la izquierda genera un parámetro de entrada para el filtro paso bajo, que va descendiendo gradualmente gracias al objeto [line] y los posteriores escalados. Esta frecuencia irá descendiendo linealmente hasta llegar a 0 en 600 milisegundos. Esta señal de baja frecuencia sirve para modular un ruido blanco, cuyos valores quedan acotados entre +1 y -1, consiguiendo así una señal con muchas variaciones pero con una oscilación en sus valores máximos que produce el efecto deseado. El filtrado paso-banda posterior sirve para suavizar el efecto del recorte introducido por el objeto [clip~] y la señal resultante se multiplica por un número que descenderá linealmente de 1 a 0 en 6 segundos para que el sonido se apague suavemente.

Finalmente, el último sonido perteneciente a este efecto es el de refracción. No todas las ondas viajan a la misma velocidad en el aire. La refracción es el efecto por el cual algunas frecuencias viajan más rápidamente que otras, normalmente por la presencia de agua en el aire o por diferencias de temperaturas. A mayor distancia, este efecto es más notable. Las frecuencias bajas viajan a menor velocidad que las altas. En el caso del trueno, normalmente, se produce a kilómetros del oyente, luego este efecto tiene una importancia considerable [33]. En la Figura 3.23 se observa la implementación final, con el nombre de “refraction”.

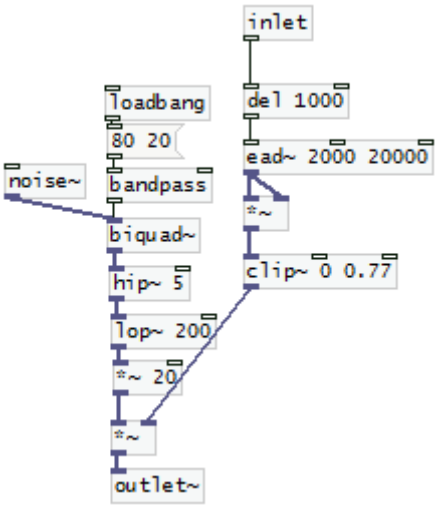


Figura 3.23: Refraction

El objeto [biquad~] configura un filtro de dos polos y dos ceros cuyos coeficientes son proporcionados por el objeto [bandpass]. Éste último recibe los dos parámetros correspondientes a la frecuencia central y al ancho de banda en el mensaje superior. El ruido blanco es pasado por este filtro, y por un paso-alto posteriormente que elimina posibles muestras de baja frecuencia que pudieran contaminar el sonido. Por último, se filtran las frecuencias altas, puesto que se trata de un sonido de baja frecuencia, y se multiplica por el resultado de la columna de la derecha. Dicha columna genera una señal con un retardo de un segundo respecto al resto del sonido para dar tiempo a los anteriores efectos a actuar. Se genera una envolvente exponencial decadente que hace que el sonido se apague lentamente, al igual que el de reverberación.

Por último, en la Figura 3.24, se muestra como se combinan los tres efectos para generar un sonido procedural de un trueno.

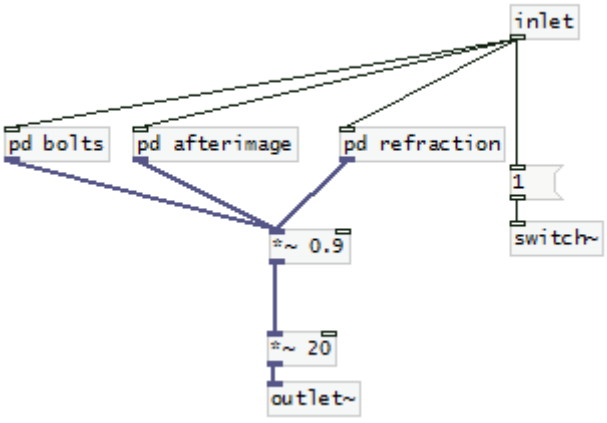


Figura 3.24: Trueno

Los tres parches toman como entrada el evento generador del trueno y, cada uno en su momento, se ponen en funcionamiento. Después, se suman y se escalan correctamente para que se produzca a un nivel de volumen coherente.

## 3.8 Pisadas

¿Qué sucede cuando un individuo camina? El cuerpo humano en reposo distribuye equitativamente el peso en ambos pies, que tienen una superficie relativamente pequeña. Existe una fuerza, una presión ejercida sobre el suelo, que es equilibrada con la fuerza recíproca del suelo hacia el cuerpo. Esto es conocido como la fuerza de respuesta del suelo, o GRF (Ground Response Force). Una superficie más pequeña significa una GRF más grande, estando de puntillas se produce más presión que en posición normal. Mientras el individuo avanza, el peso cambia de un pie a otro y la energía se expande por los músculos para llevarlo hacia adelante [33]. Existen tres fases en el proceso de un paso, ya que el pie no se apoya en el suelo plano. En la primera fase, el talón impacta con el suelo. En la segunda, el pie rueda apoyando el resto de la planta hasta llegar a los dedos. En la última fase, el individuo se impulsa hacia adelante utilizando los dedos y una serie de músculos. En la Figura 3.25 se puede ver una representación gráfica del proceso [33].

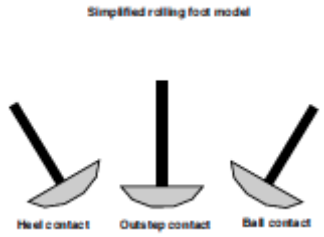


Figura 3.25: Paso

La velocidad de movimiento determina el tiempo en el que el pie está en contacto con el suelo. Cualquier patrón de GRF es comprimido o expandido cuando la velocidad del individuo cambia. La longitud de cada paso debe cambiar con la velocidad mientras el sonido de la interacción con la textura del suelo debe permanecer constante [33]. En la Figura 3.26, podemos ver la representación gráfica de varias funciones GRF según la velocidad del individuo.

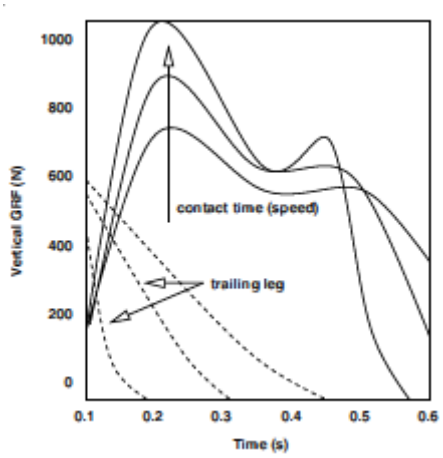


Figura 3.26: Representación de señales GRF

El modelo para generar sonidos de pisadas consiste en realizar una representación de estas funciones GRF, que servirán como señales de control, y aplicarles las propiedades frecuenciales de cuatro superficies: gravilla, madera, nieve y césped (follaje). El parche superior sirve para calcular los parámetros de las funciones en base al movimiento del personaje. Estos parámetros serán introducidos a otro parche que será el encargado de activar la textura que debe sonar en base a la posición del personaje en el mapa. Finalmente, para cada textura, hay otro parche de nivel inferior, en el cual se moldeará la señal GRF en base a las características de cada una y se aplicarán las propiedades frecuenciales, teniendo como salida el sonido final. En la Figura 3.27, vemos el parche superior.

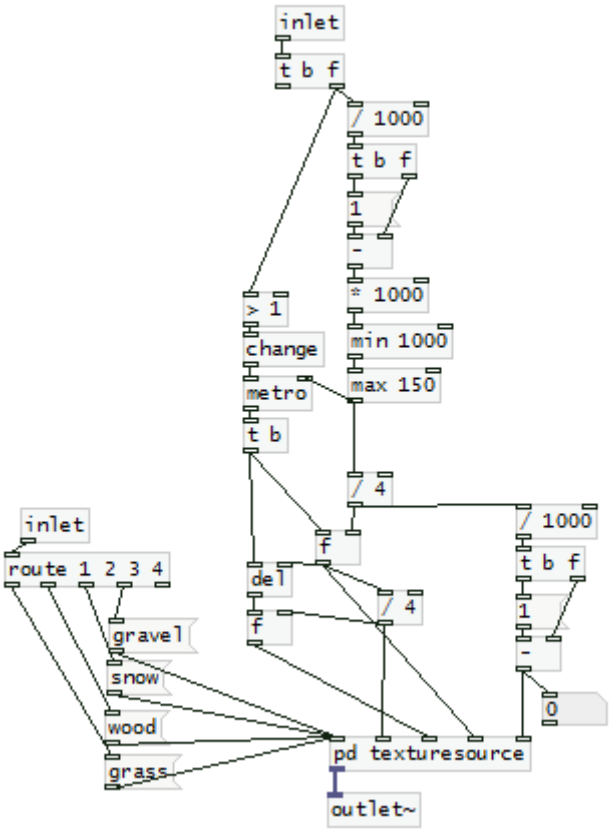


Figura 3.27: Pisadas

El parche toma como entrada el ritmo al que se está moviendo el personaje. Hay tres valores para cada uno de los tres movimientos: caminar, trotar y esprintar. Las operaciones siguientes tienen como función calcular los parámetros que son introducidos a [pd texturesource]. Hay que tener en cuenta que, estos parámetros luego son introducidos en un objeto [vline~] y posteriormente se les aplica un coseno para darle la forma de la función. Representan una medida de longitud de la señal GRF para cada una de las tres fases. En la parte izquierda podemos ver otra entrada que determinará el mensaje a transmitir al objeto para seleccionar la textura que sonará en cada momento. En la Figura 3.28 se muestra [pd texturesource].

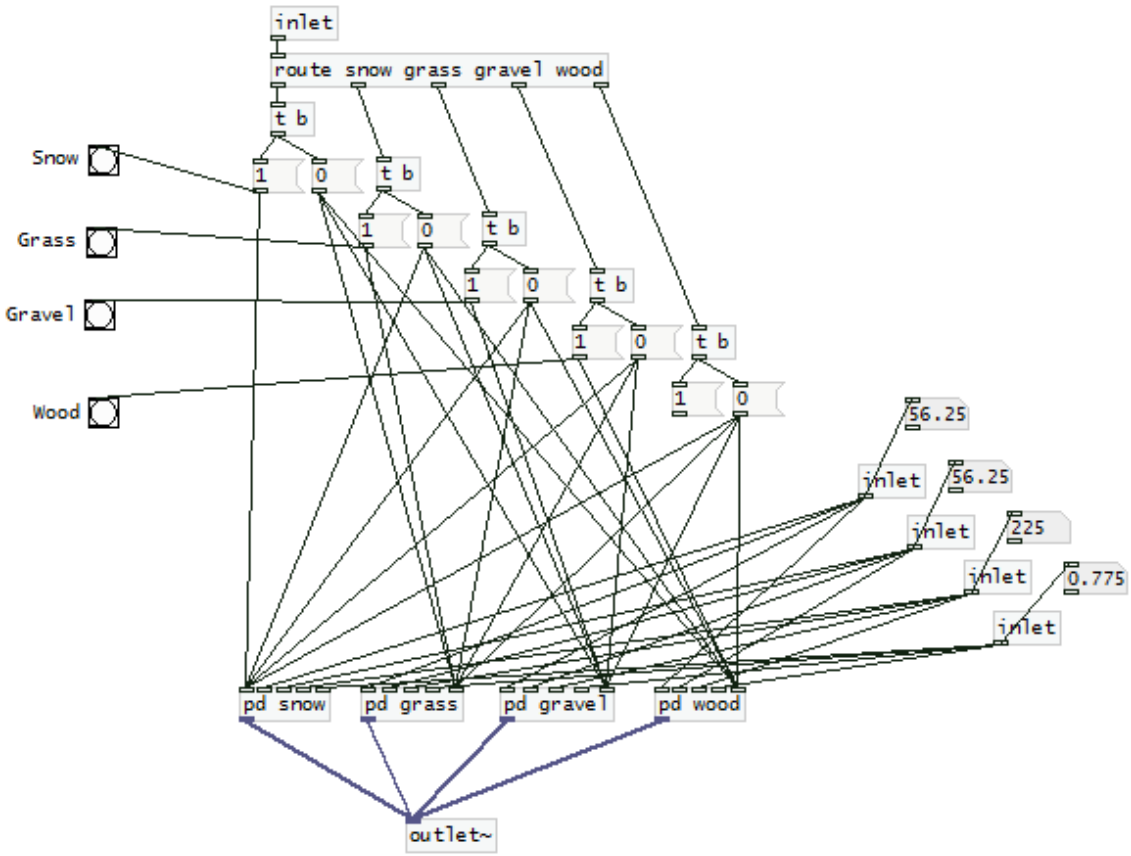


Figura 3.28: Texturesource

La entrada que recibe es el mensaje enviado, que corresponde con las palabras en inglés de las cuatro texturas presentes en la escena. Cuando recibe una de ellas, por ejemplo “snow”, el parche envía un “1” al subparche [pd snow] y un “0” al resto, permitiendo así solo una textura activa cada vez. También ejerce como puente para los parámetros de la señal GRF, trasladándolos a todos los subparches. A continuación se explican las particularidades de cada textura.

### 3.8.1 Gravilla

Se entiende como gravilla una superficie en la que hay presente tanto tierra como pequeñas piedras, que puede estar presente en paisajes naturales montañosos, caminos, etc.

Esta superficie produce un sonido muy rugoso al caminar sobre ella, producto de la fricción de la suela con la gran cantidad de pequeñas piedras bajo ella. Tomando como punto de partida una grabación digital, se extraen la forma de la señal temporal de una pisada así como su transformada al dominio de Fourier y su espectrograma para saber en qué momento qué frecuencias tienen más relevancia.

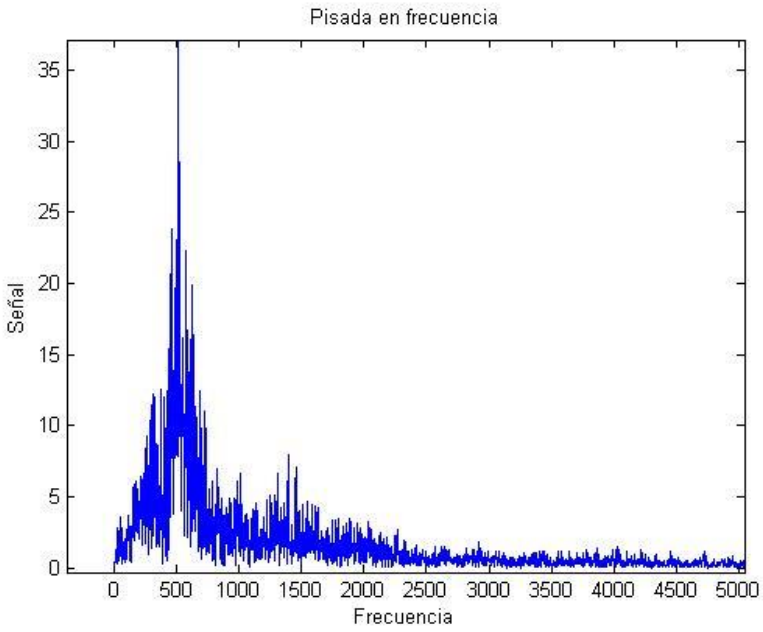


Figura 3.29: Pisada sobre gravilla en frecuencia

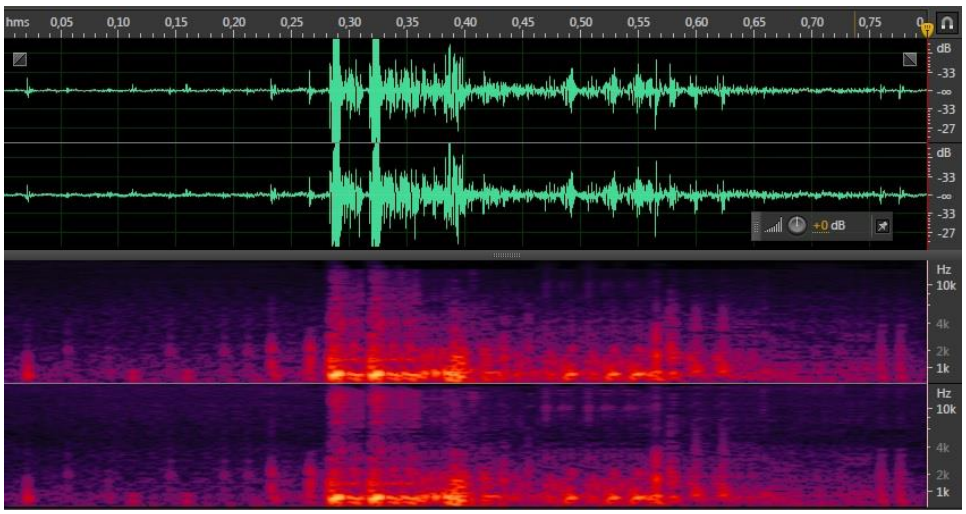


Figura 3.30: Espectrograma de una pisada en gravilla

De la representación frecuencial se puede observar como la mayor parte de la información se encuentra en una banda estrecha alrededor de 500Hz, aunque también hay un pico menor alrededor de 1500Hz. Del espectrograma se saca la conclusión de que en el momento que se posa el talón y que se impulsa con los dedos es en el que con mayor intensidad la energía se concentra en estas bandas, así que en la implementación se busca la manera de hacer coincidir estos hechos. También vemos una ligera aunque considerable frecuencia de presencias altas, lo que invita a pensar que en la implementación los filtros no deben ser demasiado estrechos. En la Figura 3.31 se puede contemplar la implementación de la textura.

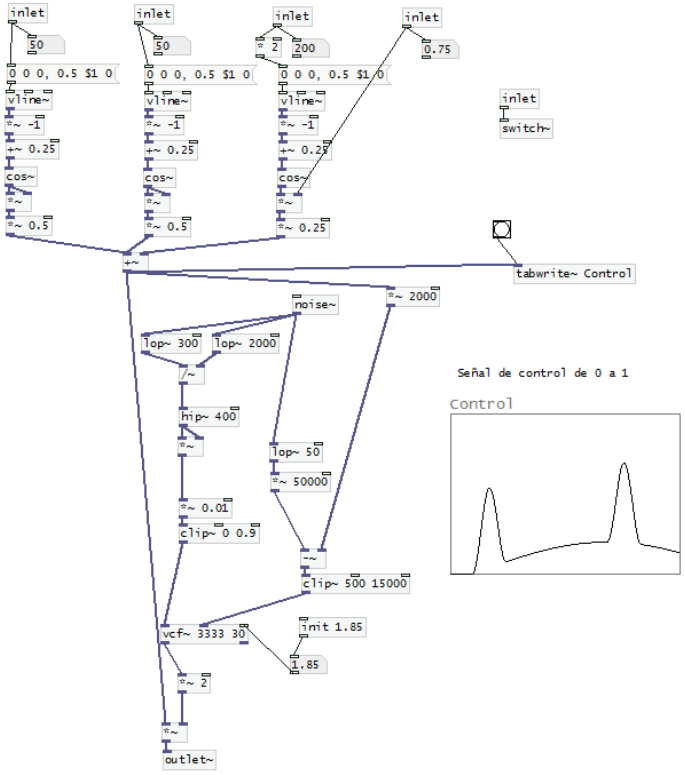


Figura 3.31: Gravilla

Lo primero a destacar en el parche es la representación de la señal GRF. En esta textura se ha dejado como en el concepto inicial, siendo dos picos pronunciados a la hora de apoyar el pie y los dedos al impulsarse, y una curva más suave para la transición entre el talón y los dedos. En la parte superior del parche, tenemos las tres columnas que reciben los parámetros del parche superior y, en base a éstos, dan forma a la señal GRF utilizando objetos [cos~]. Esta señal es modulada al final del parche con la señal ruidosa. Se toma como fuente un ruido blanco que tomará dos caminos: por una parte conformará la señal a la que se dará forma en frecuencia para satisfacer lo visto en el estudio previo, que se realizará con un filtro paso-banda controlado por tensión, y por otro lado se generará la señal que controla este filtro. La señal a filtrar será una señal de muy altas variaciones, con muchos picos, simulando la fricción entre las miles de pequeñas piedras que conforman la textura. Esto se consigue dividiendo dos señales, una con mayor presencia de altas frecuencias que otra, provocando valores muy altos cuando la que ocupa el lugar del divisor pase por cero. Esta señal se filtra paso-alto, eliminando las bajas frecuencias, y se eleva al cuadrado para hacer los valores de pico más finos. Finalmente, buscando la coherencia en el volumen de la salida, se limitan estos valores posibles entre 0 y 0.9.

Por el otro lado, se filtra paso-bajo la señal del ruido en 50Hz, esto genera una señal de muy pocas variaciones. Tras el posterior escalado, se le resta la señal de entrada multiplicada por 2000 y se acotan sus valores entre 500 y 15000. Con esto se consigue que, cuando los picos de presión son mayores (la señal de entrada toma valores entre 0 y 1), la frecuencia central del filtro tomará como valor 500Hz o valores cercanos, enfatizando así la presencia de estas frecuencias en esos momentos. Finalmente, se multiplica la señal de entrada por la señal generada y se obtiene el resultado final.

### 3.8.2 Madera

Hay muchos tipos de superficies de madera para implementar un sonido general. En el caso de esta escena, se optó por intentar recrear un sonido característico de un puente. Para ello, como en el caso de la gravilla, se partió de estudiar las características frecuenciales de una grabación. El sonido, por otra parte, resulta muy seco, no tan rugoso como el de la gravilla. En las Figuras 3.32, 3.33 y 3.34 tenemos las representaciones de una pisada en frecuencia, el espectrograma y la envolvente temporal de una secuencia de pisadas.

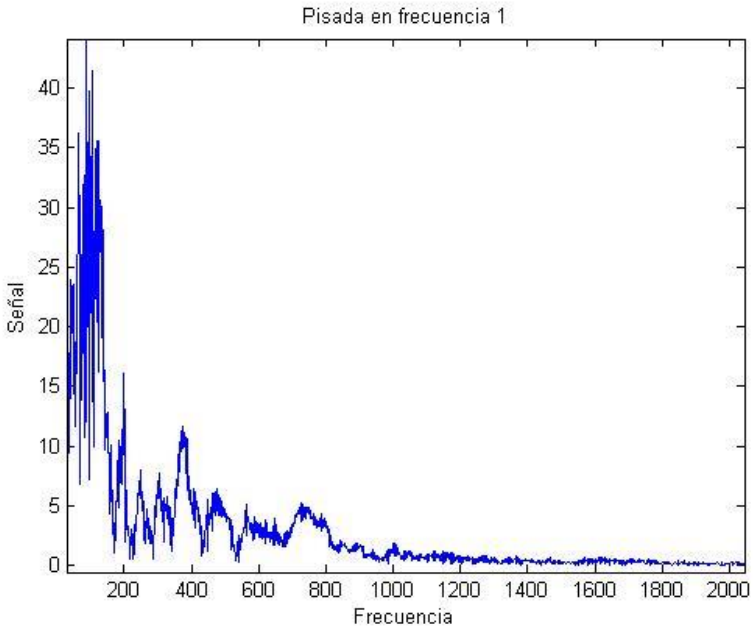


Figura 3.32: Pisada sobre madera en frecuencia

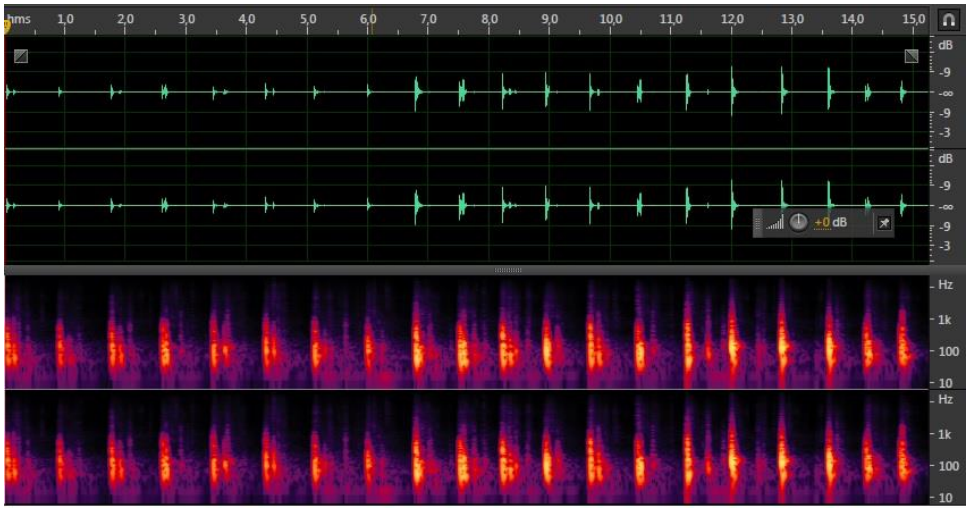


Figura 3.33: Espectrograma de una cadena de pisadas sobre madera

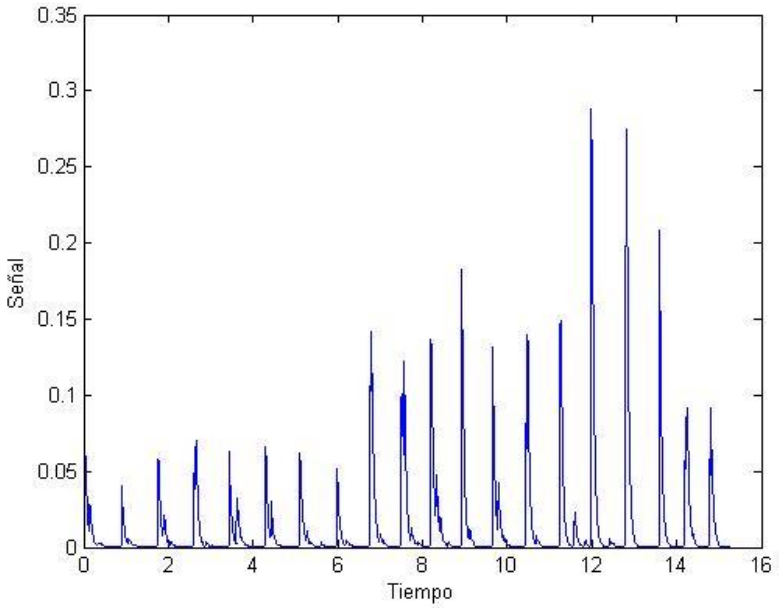


Figura 3.34: Envolvente temporal de una cadena de pisadas sobre madera

En la representación frecuencial, observamos unos picos muy marcados entre 0 y 200Hz, generalmente con las componentes más definidas, más agrupadas alrededor de un nivel concreto. Se observan valores máximos alrededor de 40, 60, 80 o 120 Hz aproximadamente. Por otra parte, en el espectrograma se observa que en los momentos de presión, la banda por debajo de 1KHz es la que cobra más importancia, además de ver muy marcado el salto entre los dos picos de presión con el pequeño espacio en negro que separa en dos cada paso. Esto se refrenda en la observación de la envolvente temporal de la cadena de pisadas, donde observamos picos muy marcados que se manifiestan en un sonido seco. También se aprecia la aparición de otras frecuencias, resultantes de la fricción entre la suela y la superficie. En la Figura 3.35 se puede observar el parche.

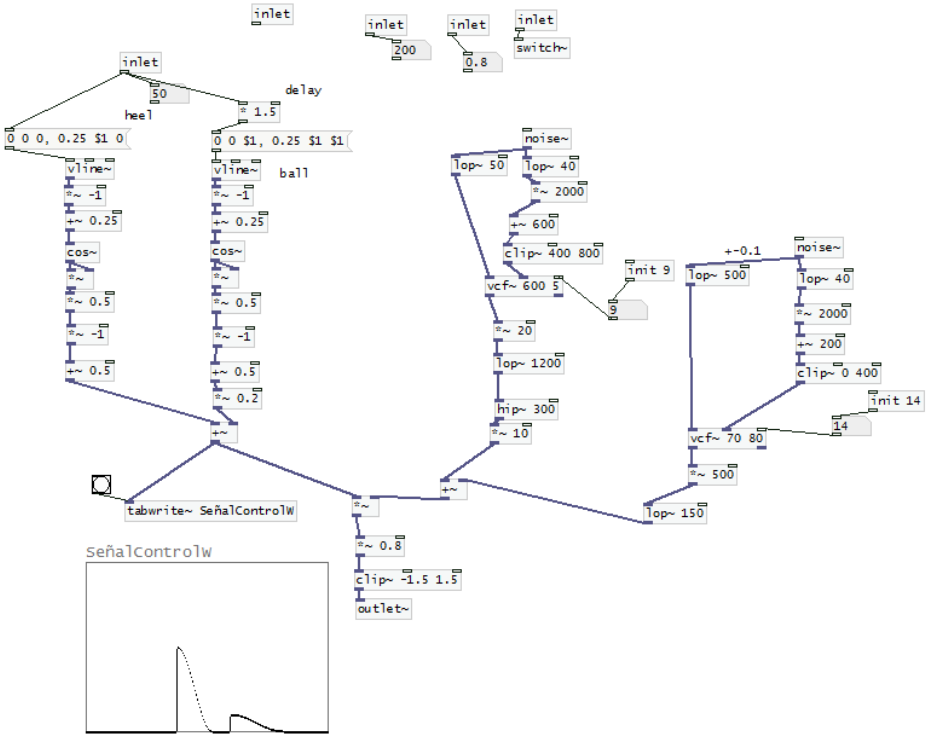


Figura 3.35: Madera

Lo primero que llama la atención en este parche es la modificación de la señal GRF. Tras lo visto previamente en la envolvente temporal y el espectrograma, había que modificar esta señal para conseguir el efecto de sequedad en el contacto. Se podía observar en la envolvente temporal de la cadena de pasos que la señal sube abruptamente, y no gradualmente como en la gravilla, así pues, se ha moldeado la señal GRF para que así fuera, prescindiendo también de la presión ejercida por la planta al pasar del talón a los dedos, considerando que el sonido producido entonces era despreciable frente al de los picos. Esto se genera en el parche con las dos columnas de la izquierda, donde en el parche de la gravilla había tres. En las columnas de la derecha, se generan la señal que multiplicará después a la GRF y que hay que moldear frecuencialmente. Se escogen dos rangos a representar: de 0 a 400Hz (la de más a la derecha), que será el que tenga más relevancia., y de 400 a 800Hz (la más centrada) para rellenar el espectro y que tendrá menor energía que las bajas componentes. El diseño para el rango de 0 a 400Hz consiste en un filtro paso-banda controlado por tensión, con un valor del factor de calidad alto, que hace que los picos queden muy destacados en el espectro al ser el filtro muy estrecho. Por otra parte, en el rango de 400 a 800Hz, el valor del factor de calidad es más bajo, luego el filtro es más ancho y permite la aparición de más componentes. Posteriormente, se suman ambas señales y se multiplican a la señal GRF para generar el sonido final.

### 3.8.3 Nieve

Esta superficie es, probablemente, la que tiene características más singulares de las cuatro. En cuanto un individuo apoya su talón para iniciar un paso sobre una capa de nieve, se resquebraja y se comprime, produciendo un ruido parecido al del hielo rompiéndose, pero

suavizado al tratarse de una textura más esponjosa. En las Figuras 3.36, 3.37 y 3.38 se representan una pisada sobre nieve en frecuencia, el espectrograma, y la envolvente temporal de una cadena de pasos.

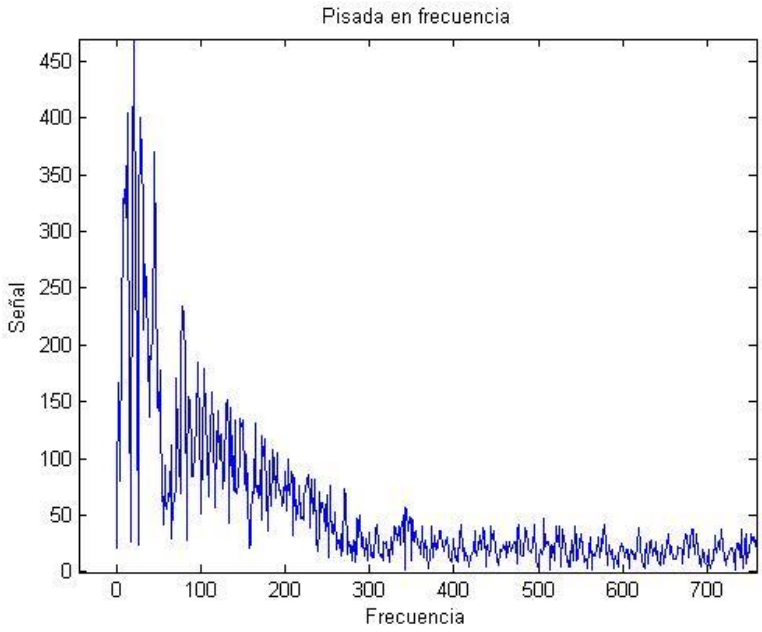


Figura 3.36: Pisada sobre nieve en frecuencia

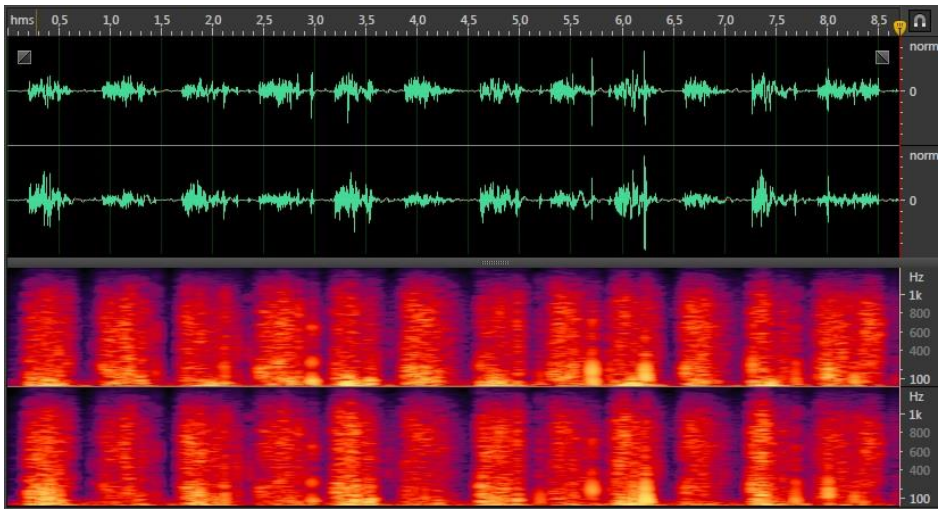


Figura 3.37: Espectrograma de una cadena de pasos sobre nieve

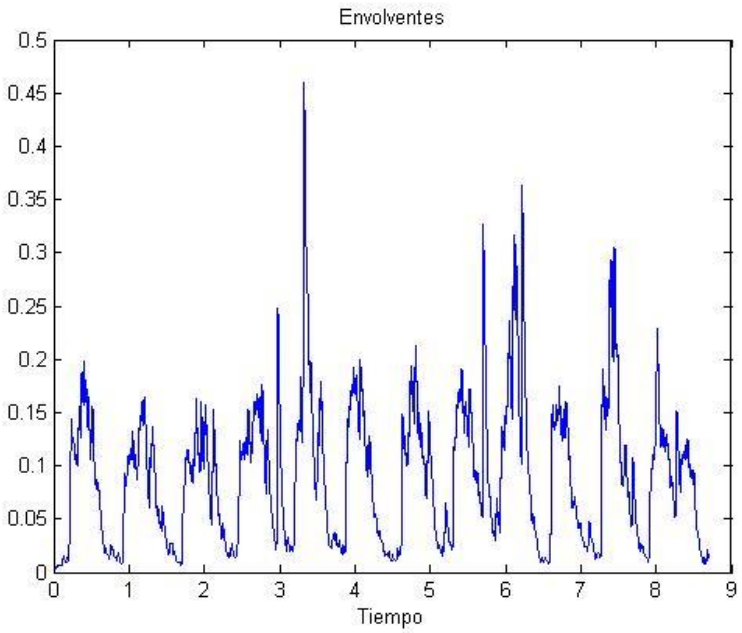


Figura 3.38: Envolvente en el tiempo de una cadena de pasos sobre nieve

Tanto en la representación en frecuencia, como, especialmente, en el espectrograma, se puede apreciar que la mayor parte de la información se encuentra en la banda por debajo de 1KHz. Es palpable que, en el momento del primer contacto, la presencia de las bajas frecuencias, del orden de las decenas de Hz, es la más relevante. Además, esta distribución es prácticamente invariable a lo largo de la pisada.

Otra observación a destacar de estas figuras es que la envolvente pierde su forma característica de dos picos pronunciados y una curva central. Se percibe una forma más redondeada, la energía total de la pisada distribuida más equilibradamente. La señal GRF original como señal de control, al igual que en el caso de la madera, no sirve en este modelo. A continuación se muestra en la Figura 3.39 el parche resultante y se explica cómo las conclusiones del estudio previo son plasmadas en él.

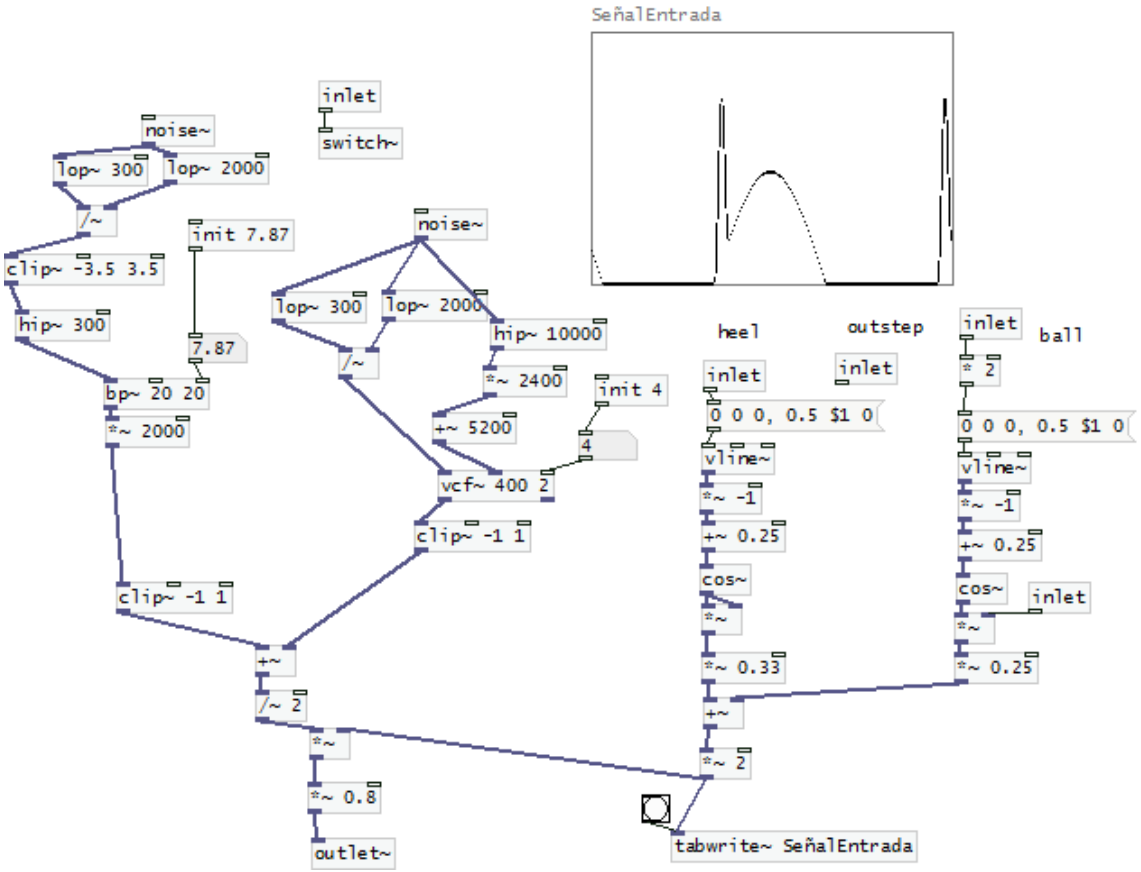


Figura 3.39: Nieve

Como se puede observar en la Figura 3.39, la señal de control en este caso consiste en un pico pronunciado al principio, para generar el crujido del primer contacto del talón con la superficie, momento en el que se concentra más energía en toda la pisada, y una segunda curva más suave, aunque cercana en amplitud, para el resto. Para implementar esto se ha prescindido del segundo pico de la señal GRF original y se ha moldeado la fase intermedia adecuadamente a las características de este sonido, para conseguir señales de salida de formas similares a la envolvente de la Figura 3.38.

El procedimiento a seguir en el resto del parche es generar inicialmente una señal de muchas variaciones dividiendo dos señales (una de bajas frecuencias y otra con presencia de frecuencias más altas que tenga pasos por 0) y filtrándolas paso-alto para simular el efecto del crujido de la nieve al romperse y compactarse debajo de la suela. Posteriormente, esta señal se pasa por un filtro paso-banda centrado en 20Hz con un valor del factor de calidad alto que haga énfasis en sus frecuencias cercanas. A esta señal se le suma una segunda que se encargará de llenar el resto del espectro con un procedimiento similar, pero con una frecuencia central variable adaptada al rango de interés (del orden de cientos de Hz) y con un valor para el factor de calidad más bajo que permita la presencia de un mayor rango de frecuencias. Tras esto, únicamente queda multiplicarle la señal de control y escalarla propiamente para producir un volumen de salida coherente.

### 3.8.4 Césped o follaje

Es muy complicado definir un sonido concreto y global para el césped o follaje. Es altamente dependiente del tipo de vegetación, de la humedad en el ambiente y de otros incontables factores  muy difíciles de controlar. Por ello se ha optado por intentar generar un sonido similar al producido cuando se camina por un paisaje montañoso y boscoso. Aunque el campo de la generación de pisadas en el audio procedural todavía tiene muchas fronteras que cruzar y mucho desarrollo potencial, es quizá esta superficie en la que más se puede innovar y en la que más variedad realista se puede recrear.

Comenzamos con el análisis frecuencial de una pisada, que podemos contemplar en la Figura 3.40.

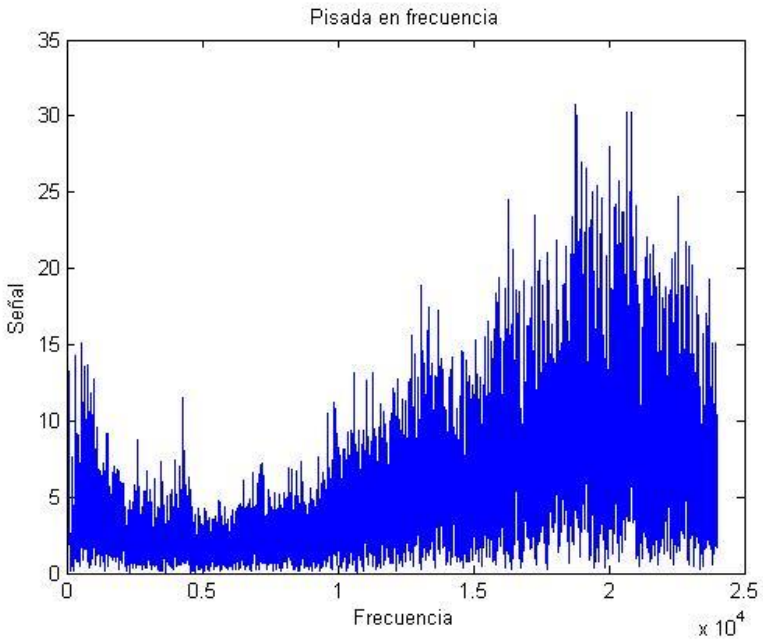


Figura 3.40: Pisada sobre césped en frecuencia

Llama la atención a primera vista que, a diferencia del resto de sonidos de pisadas representados hasta ahora, la mayoría de la energía en una pisada sobre un terreno con césped o vegetación se concentra en las altas frecuencias. El tener un espectro tan rico hace recomendable estudiar su espectrograma por partes. Sin embargo, como se puede comprobar en la Figura 3.41, esta vez la forma de la envolvente sí que corresponde a los dos picos junto a la curva más suave de la fase intermedia. Además, en el espectrograma de la Figura 3.42 también se puede ver que la presencia de las altas frecuencias es más importante en los momentos de mayor energía del talón y los dedos.

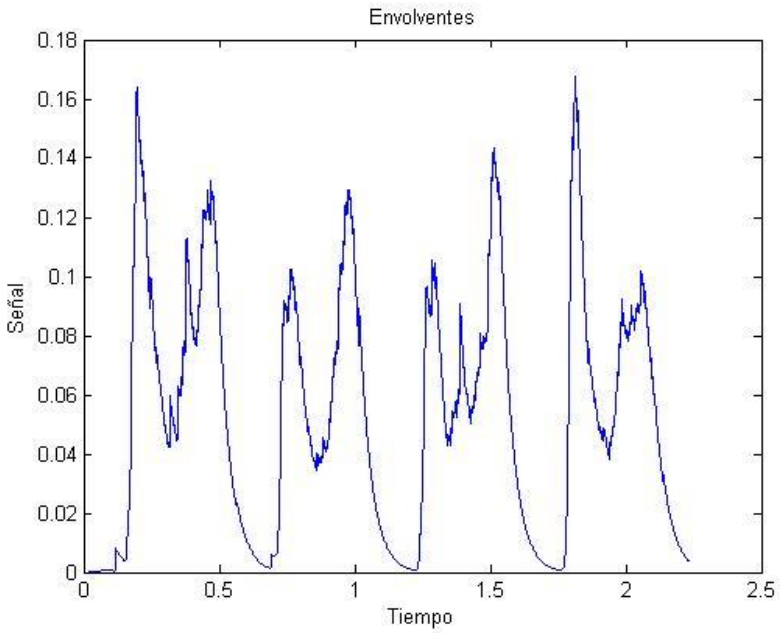


Figura 3.41: Envolvente en el tiempo de pasos sobre césped

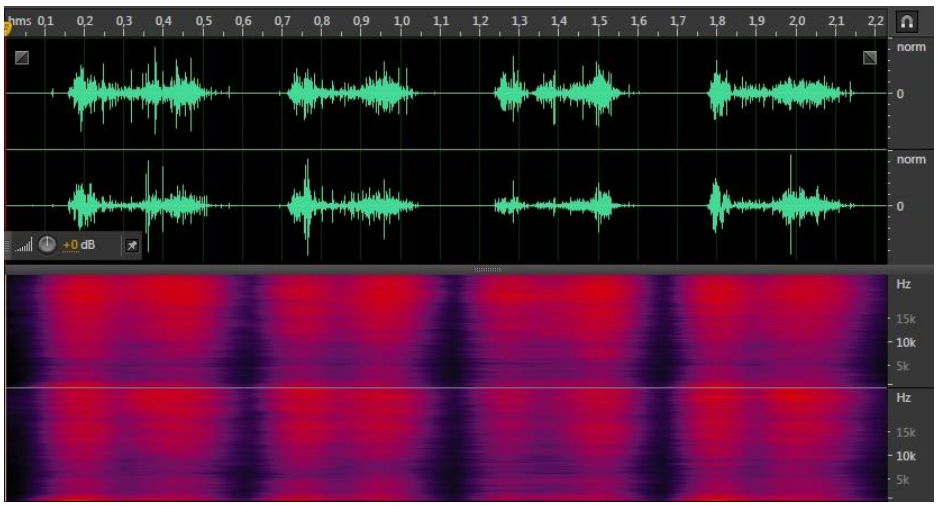


Figura 3.42: Espectrograma de una cadena de pasos sobre césped

La estrategia a seguir en la implementación consistió en generar por separado cada una de las bandas influyentes en el sonido y sumarlas para luego aplicarlas a la señal GRF entrante. Se puede ver dicha implementación en la Figura 3.43.

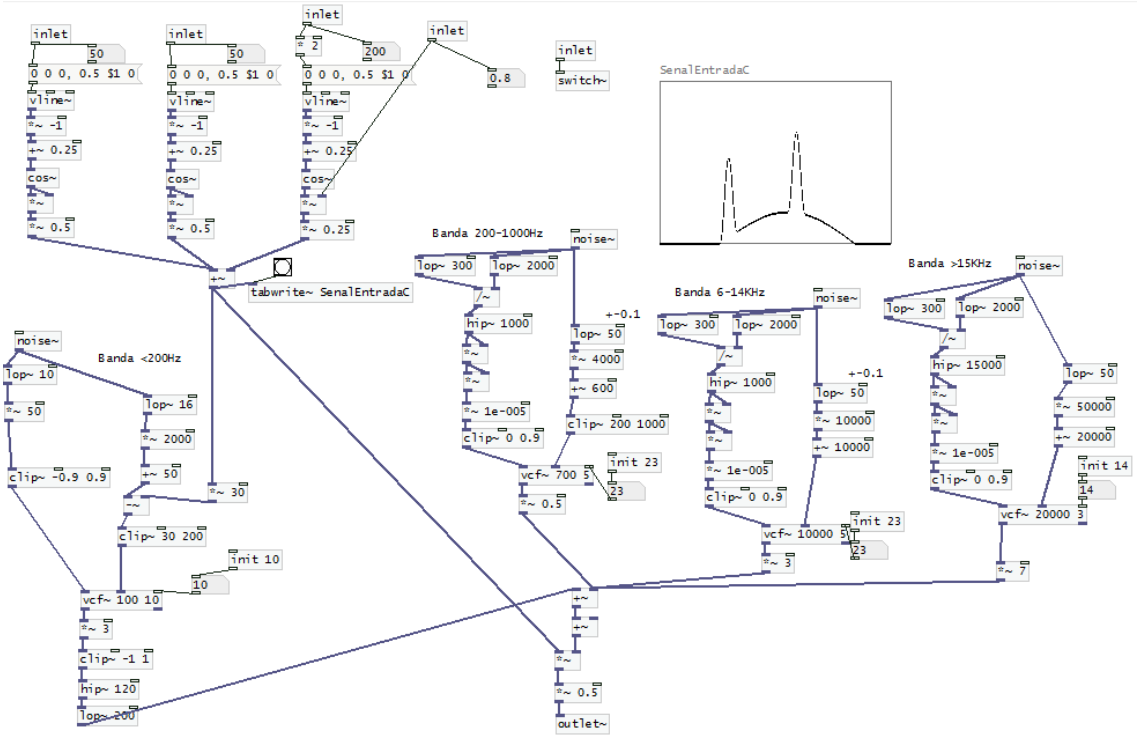


Figura 3.43: Césped/Follaje

Como puede apreciarse, las bandas se han separado en cuatro: 30-200Hz, 200-1000Hz, 6-14KHz y mayores de 15KHz. La banda menor de 200Hz tiene una fuerte presencia en los picos, resultado del golpeo del talón y la punta de la suela sobre un terreno que no es duro. Se ha utilizado la señal de entrada para que, en esta banda, cuando los momentos de presión sean mayores, los valores de la frecuencia central del filtro sean más graves. Al ser un sonido profundo y no tan rugoso, se ha utilizado simplemente una señal de baja frecuencia, a diferencia del resto de bandas.

En las bandas de alta frecuencia sí que se ha utilizado la técnica de la división de dos señales para crear los picos generados por el crujir de ramas y hojas secas bajo el pie. Estas tres columnas son idénticas y tan solo cambian los valores de las frecuencias introducidos a los filtros, así como los valores de los factores de calidad, ajustados empíricamente. En el sonido producido se perciben crujidos secos, característicos de ramas y de hojas secas, de paisajes boscosos y montañosos.

## 3.9 Mezclador

Todos estos efectos anteriormente comentados han sido unificados en un gran parche para que puedan sonar simultáneamente y poder monitorizar la comunicación entre PD y la escena creada con Unity 3D. En la Figura 3.44 se puede observar el parche.

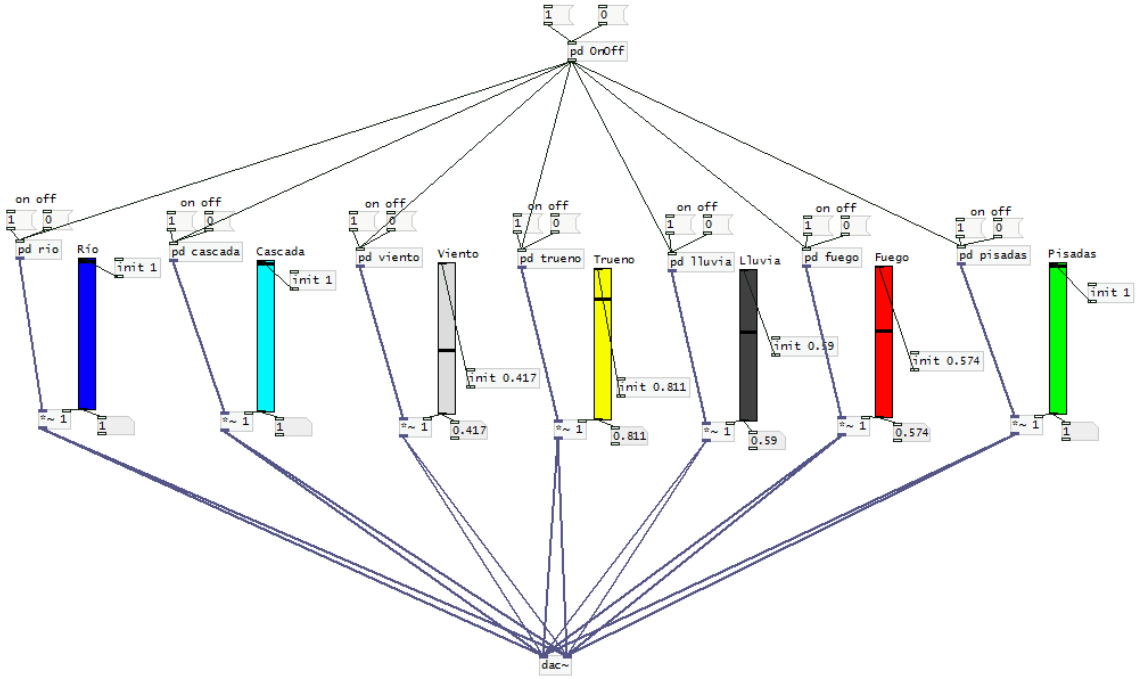


Figura 3.44: Mezclador

El parche basa su funcionamiento en unos Vslider que aplican un control de ganancia a las salidas de los efectos, para poder adecuar sus niveles de volumen a los del resto. En otras palabras, el parche actúa como un mezclador con el que tener un control sobre las ganancias de las señales. Cada efecto puede apagarse o encenderse manualmente y, en la parte superior, se encuentra un subparche que enciende o apaga todos los efectos simultáneamente en base a si la escena en Unity se encuentra en ejecución o no.

# Capítulo 4: Escena en Unity y comunicación OSC

## 4.1 Escena

Se ha utilizado el entorno de desarrollo Unity 3D para recrear una escena montañosa y boscosa, en la que se pueda provocar una interactuación entre sus elementos que den virtualmente lugar al desencadenamiento de los efectos de sonido implementados.

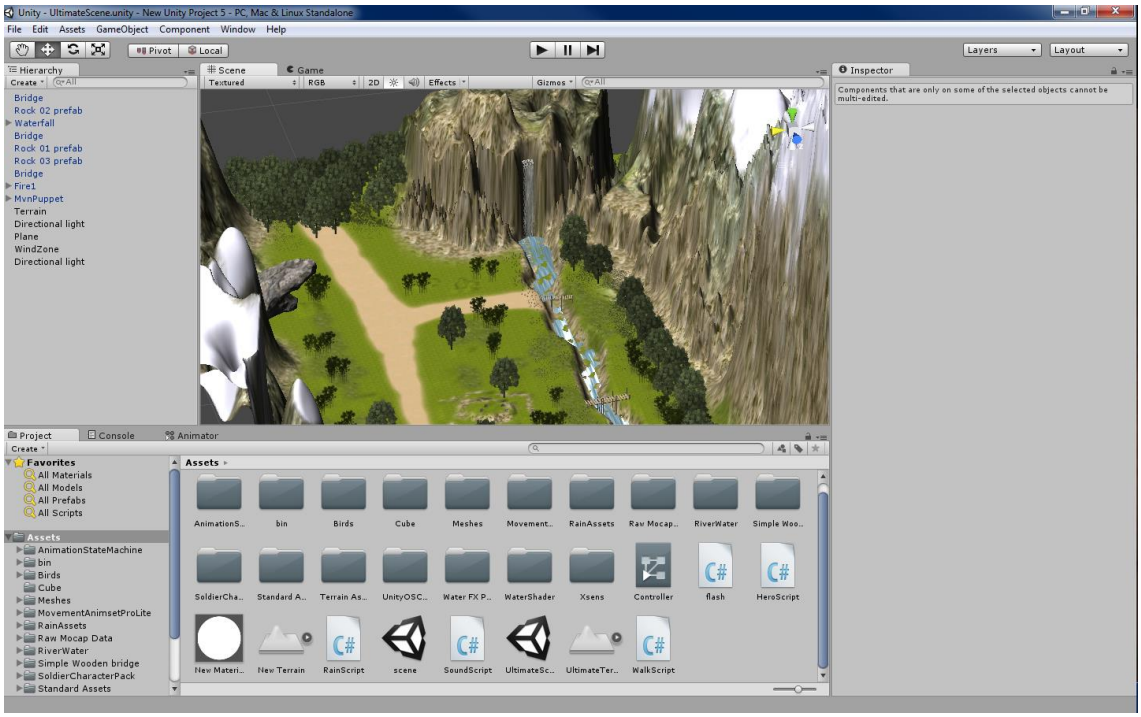


Figura 4.1: Entorno de Unity 3D

El entorno es altamente personalizable, con ventanas a situar en cualquiera de las zonas de la pantalla, y altamente versátil. En la Figura 4.1 se observa que la pantalla de trabajo está configurada de tal manera que en el conjunto de ventanas central se puede ver alternativamente la escena y la cámara principal (si cambiamos de pestaña). A la izquierda tenemos la jerarquía, en la que aparecen todos los objetos que hemos ido añadiendo. Se llama jerarquía porque, verdaderamente, los objetos pueden tener diversos elementos bajo su influencia. Estos elementos pueden ser las distintas partes por las que están configurados, u otros complementos que influyen ellos, como cámaras u otros objetos. A la derecha tenemos el inspector, donde podemos modificar las características de los objetos y añadirle elementos que interactúen con estas características, en forma de componentes prediseñados o códigos en C# o Javascript. En la parte inferior, se muestran las carpetas incluidas en el proyecto, de donde se pueden tomar los modelos, objetos o efectos que se quieren incluir en la escena. Cambiando de pestaña, se puede visar la consola, donde el programa muestra información durante la ejecución en relación a los scripts que está corriendo, interacciones entre elementos, o errores que pudiera haber en la construcción de la escena.

Unity 3D cuenta con un editor de terrenos que ha sido utilizado para crear la escena, el cual permite moldear un paisaje al gusto, permitiendo introducir valles, mesetas o montañas.

El editor también cuenta con herramientas para introducir vegetación capaz de interactuar con el viento, y texturas, lo que resulta útil para dar soporte a los efectos de pisadas.

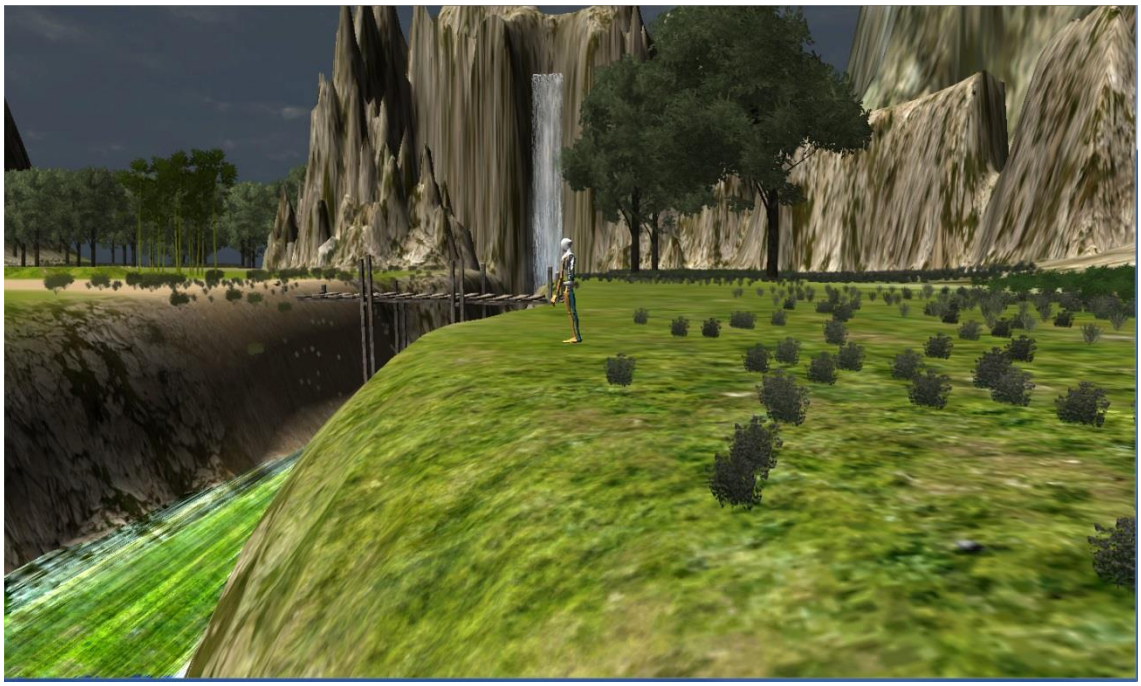

Figura 4.2: Escena y personaje principal

En la Figura 4.2 podemos apreciar algunos de los elementos principales de la escena. Se ha construido un paisaje montañoso, que se puede identificar con un valle, en el que se encuentra una cascada que da lugar a un río. Hay varios puentes de madera para cruzarlo. Existen zonas con árboles y arbustos, así como terreno en el que predomina el follaje y un camino de gravilla. Existe otra zona con nieve, pero se encuentra en lo alto de las montañas y no es accesible si no se sitúa al personaje allí desde un principio. Se ha situado una hoguera en un lugar resguardado y es posible generar lluvia y truenos. Se ha pretendido simular un día tormentoso, de manera que puedan estar presentes los efectos climatológicos implementados.

## 4.2 Personaje

Se ha escogido un modelo humanoide para actuar como personaje principal. En la Figura 4.3 podemos ver su aspecto y, junto él, el dibujo que indica donde está situada la cámara principal inicialmente.

Figura 4.3: Personaje

Se trata de un diseño muy simple, pues, siguiendo el ejemplo de las grabaciones de efectos, se ha optado por no invertir dinero alguno en complementos para el desarrollo del proyecto, por tener un carácter didáctico y ningún ánimo de exhibición o lucro.

A la hora de animar el modelo, se ha diseñado una máquina de estados que se puede observar en la Figura 4.4 a través de los cuales el personaje se irá desplazando dependiendo de las entradas que le lleguen por el teclado.

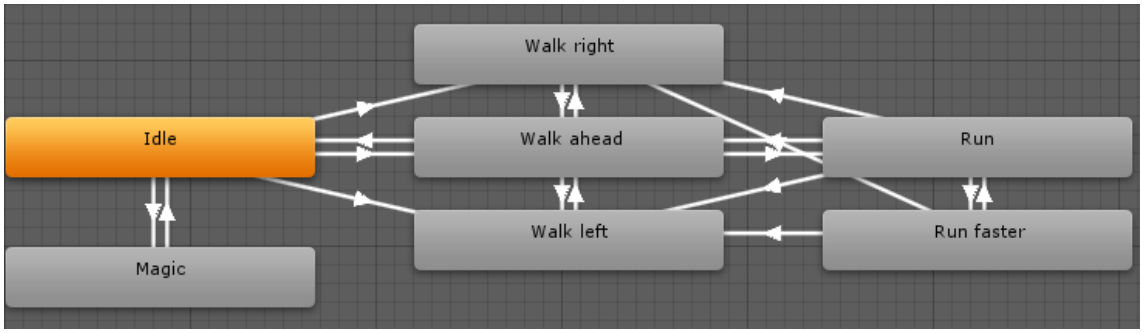


Figura 4.4: Máquina de estados para las animaciones

Esta máquina de estados, junto a los códigos incluidos en el Anexo A, son los encargados de dar vida al personaje. Cuenta con un estado inicial, el único en naranja, “Idle” (en inglés, reposo). Desde este estado es posible avanzar con la tecla “W” a “Walk ahead” (caminar hacia adelante), con la tecla “D” hacia “Walk right” (girar a la derecha), con la “A” a “Walk left”(girar a la izquierda) o con la tecla “R” o “T” para provocar lluvia y truenos respectivamente a “Magic” (estado que lleva asociado una animación que simula la pulsación de un botón). Si se mantiene pulsada la tecla “W” se accede automáticamente al estado “Run”, que lleva asociada una animación de trote. Desde este estado, es posible volver a “Walk ahead” si, a la vez que se pulsa la letra “W”, se pulsa también la letra “C”. También es posible esprintar, a la vez que “W”, si se pulsa la letra “F” saltará al estado “Run faster”. En las tres velocidades es posible girar pulsando “D” o “A” para derecha e izquierda respectivamente.

## 4.3 Comunicación por mensajes OSC

### 4.3.1. Conceptos

El protocolo OSC (OpenSound Control) es un protocolo desarrollado para la comunicación entre ordenadores, sintetizadores y dispositivos multimedia en general abierto, basado en mensajes y transparente al medio en el que se propaga.

La unidad de transmisión para el protocolo es denominada “paquete OSC”. Cualquier aplicación que manda un paquete OSC es un “cliente OSC” y cualquier aplicación que recibe un paquete OSC es un “servidor OSC”. Un paquete OSC consiste en una cadena de números binarios y su tamaño es siempre múltiplo de 4 bytes. La capa inferior que entrega un paquete OSC es responsable de entregar tanto el contenido como el tamaño a la aplicación. Un paquete puede ser naturalmente representado por un datagrama en un protocolo de red como UDP [34].

De esta manera, en este proyecto Unity 3D actuará como cliente y PD como servidor como se muestra en la Figura 4.5.


Figura 4.5: Esquema del proyecto

En la escena creada en Unity 3D, el personaje principal interactúa con el entorno, provocando o simplemente escuchando efectos de sonido a su alrededor debidos a los diversos fenómenos. Estos fenómenos generan vía código unos mensajes OSC, que son escuchados por PD y reproduce el sonido correspondiente con las características que en ese momento sean requeridas.

### 4.3.1 Implementación en Pure Data

Para dar soporte a la comunicación vía mensajes OSC, en PD se ha utilizado la implementación “mrpeach”, una librería que debe ser importada para utilizar sus objetos característicos. En la Figura 4.6 tenemos un ejemplo de uno de los parches utilizados para recibir mensajes OSC desde Unity.

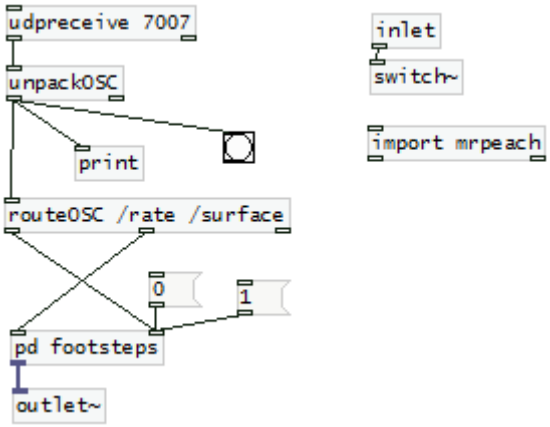


Figura 4.6: Parche para la recepción de mensajes OSC

El parche de la Figura 4.6 es utilizado para la recepción de mensajes OSC para el efecto de las pisadas. Se encuentra en el nivel inmediatamente inferior al Mezclador. Este parche recibe como entrada un mensaje en forma de "1" si tanto el sistema como el efecto está activo y un "0" si alguno de los dos no lo está. La implementación es ésta porque hacía posible retocar los efectos de sonido con la ejecución de la escena en Unity parada en la fase de desarrollo, puesto que es posible activar manualmente el parche.

Unity 3D envía mensajes a los puertos del equipo, mientras que PD escucha esos puertos a la espera de recibirlos. En el caso del efecto de las pisadas, el puerto que se utiliza para mandar y recibir los mensajes es el 7007. Una vez recibido, el mensaje es descomprimido por el objeto [unpackOSC]. Tras esto, el valor o los valores obtenidos son trasladados al parche que recrea el efecto de sonido como entrada. La señal de salida será la señal de sonido.

Cada efecto de sonido tiene sus parámetros particulares de control, su puerto de escucha, y las teclas que lo hacen funcionar, los cuales son explicados a continuación:

Río

- Puerto: 7006
- Parámetros:
  - /river valor de amplitud de la señal de sonido calculado en base a la distancia euclídea entre el personaje y la posición del río.
- Teclas: NA

Cascada

- Puerto: 7003
- Parámetros:
  - /waterfall valor de amplitud de la señal de sonido calculado en base a la distancia euclídea entre el personaje y la posición de la cascada.
- Teclas: NA

Viento

- Puerto: 7004
- Parámetros:
  - /windon parámetro binario que pone en funcionamiento el efecto
  - /velocity determina la velocidad del viento entre brisa y vendaval
- Teclas:
  - “V” para alternar entre las dos velocidades

Trueno

- Parámetros:
  - /thunder parámetro que actúa como “bang” y que únicamente puede valer “1” para activar el sonido del trueno
- Puerto: 7002
- Teclas:
  - “T” para activar

Lluvia

- Parámetros:
  - /rainon “bang” que enciende el efecto de lluvia
  - /rainoff “bang” que apaga el efecto de lluvia. Son necesarios dos porque se tratan de un efecto continuo
- Puerto: 7001
- Teclas:
  - “R” para activar y desactivar

Fuego

- Parámetros:
  - /fire valor de amplitud de la señal de sonido calculado en base a la distancia euclídea entre el personaje y la posición de la hoguera.
- Puerto: 7005
- Teclas: NA

Pisadas

- Parámetros:
  - /rate sirve como parámetro que marca el ritmo de pisada, es decir, si el personaje camina, trota o esprinta.
  - /surface indica en qué superficie se encuentra.
- Puerto: 7007
- Teclas:
  - “W” para avanzar.
  - “A” para girar a la izquierda.
  - “D” para girar a la derecha.

- “C”, mientras se avanza, para caminar.
- “F”, mientras se avanza, para esprintar.

Por último, se ha utilizado el puerto 7008 para recibir un parámetro llamado /state que indica si la escena en Unity está en ejecución o no, lo que sirve para activar y desactivar el Mezclador. En el Anexo A, junto al resto de códigos implementados en Unity, se encuentran las funciones realizadas en C# para enviar los mensajes OSC a PD. Para ello se ha utilizado la librería “UnityOSC-master” de Jorge García [35].

# Capítulo 5: Conclusiones finales y líneas futuras

A lo largo del desarrollo del proyecto, se ha producido una inmersión en el mundo del audio para videojuegos en general, y en el del audio procedural en particular, intentando comprender las ventajas y limitaciones de cada disciplina, así como alcanzar un cierto punto de comprensión y habilidad en la que nos atañe, que es el audio sintético y procedural.

Aunque no se ha profundizado demasiado en el uso de las matemáticas y la física para la implementación de los efectos de sonido, se pueden considerar algunos de los resultados objetivamente como satisfactorios. Se partió como base, principalmente, del trabajo de Andy Farnell en la materia, evolucionándolo y optimizándolo en ciertos casos, siguiendo un camino propio en otros, llegando a tener suficientes efectos como para suministrar sonido a una escena montañosa o boscosa casual en un videojuego.

Si bien es cierto que algunos efectos como el de fuego ya estaban bastante desarrollados y solamente han sido necesarios algunos retoques para su adecuación a la escena, se puede decir objetivamente que ha habido considerables avances en otros como las pisadas o el río, efectos que se encontraban en un estado muy prematuro para ser considerados satisfactorios. El profundo estudio frecuencial llevado a cabo, así como la inclusión de señales GFR variables según la superficie en el caso de las pisadas, y la utilización de las posibilidades de PD para poder plasmar las conclusiones teóricas en el sonido final han dirigido hacia los resultados mostrados en la demostración.

En cuanto a la escena, como se ha mencionado previamente en el documento, se han utilizado las herramientas que incluye la versión gratuita de Unity 3D. Aun siendo la versión gratuita, Unity 3D es una herramienta muy potente para crear videojuegos o animaciones, con multitud de complementos que han sido muy útiles a la hora de integrar la escena con los efectos de sonido, así como una amplia comunidad de desarrolladores que han dado respuesta a todas las preguntas que le iban surgiendo al autor, puesto que se trataba de su primer contacto con una herramienta de este tipo. De hecho, Unity 3D constituye el “software development kit” (SDK) por defecto para una plataforma de primera línea como es Wii U. El hecho de utilizar la versión gratuita hace que visualmente sea algo limitada, luciendo peor que la mayoría de videojuegos modernos. No obstante, el objetivo del proyecto era utilizar Unity para dar un soporte visual a los efectos de sonido implementados, que constituyen el grueso del trabajo, aunque fuera de forma más básica.

Como conclusiones finales a la inmersión en el mundo del audio procedural, puede decirse que esta disciplina, al menos a medio plazo y con el nivel de tecnología actual, no va a sustituir completamente al audio pregrabado. Hay efectos, como la voz humana en una conversación a la que hay que dotarla de dramatismo, que no se van a poder simular artificialmente, y otros muchos en los que será muy complicado obtener resultados verdaderamente buenos. Se han podido comprobar sus ventajas y desventajas, desde el ahorro de memoria y recursos, la sensación de estar ante algo vivo al tener unos efectos dinámicos, y el ahorro de tiempo y esfuerzo que supone recrear esos efectos frente a la grabación al resultado estético final, que (casi) nunca será equiparable al de uno real.

Sin embargo, el audio procedural sí que puede cumplir un papel complementario al audio pregrabado. Durante la ejecución de la escena se ha observado como el resultado estético final de los efectos ya mezclados, y recibiendo también información visual, mejora perceptualmente. En la fase de desarrollo ha sido complicado conseguir una satisfacción plena con un efecto de sonido, escuchándolo repetidamente sin ningún tipo de estímulo visual y sin ningún otro efecto con los que en la implementación final va a poder escucharse. Sin embargo, al poner todo junto, el resultado final mejora, la percepción humana también entra en juego. El audio procedural es interesante para recrear sonidos continuos como la lluvia, el sonido de una hoguera, o de un río, que ocuparían una gran cantidad de memoria en caso de ser pregrabados. Además, gracias al dinamismo de la implementación, no suenan repetitivos, puesto que la grabación termina y debe volver a reproducirse en algún momento, mientras que los elementos de aleatoriedad evitan los bucles. Los efectos de pisadas sobre superficies sobre las que se puede conseguir un resultado suficientemente bueno también son útiles. Ninguna pisada, dentro de unos ciertos límites, suena igual, lo que aumenta la sensación de inmersión en el jugador.

Las posibilidades futuras del audio procedural son muy grandes, puesto que, como se ha mencionado anteriormente, se trata de una disciplina joven y sin demasiado trabajo y estudios previos. La mayor parte de la información al respecto y de la que se ha obtenido la mayoría de la documentación para este proyecto se concentra en la página web “procedural-audio.com”. El futuro y su definitiva inclusión en el mundo de los videojuegos pasan por la formación de los ingenieros de sonido en la materia.

En cuanto al trabajo aquí realizado, siempre se puede profundizar más en el desarrollo de los efectos. Una utilización más profunda de métodos matemáticos añadidos a una investigación más específica de cómo cada efecto es producido verdaderamente, más allá de estudios espectrales, podría llevar a otro nivel el resultado final. Modelados físicos y métodos granulares podrían ser utilizados para mejorar los resultados aquí obtenidos. Finalmente, la completa integración de alguna manera del motor de sonido implementado con la escena desarrollada en Unity, de forma que conformen un elemento indivisible, sería recomendable mirando hacia futuras exhibiciones o comercializaciones.

# Anexo A: Scripts en C# para Unity 3D

## Sound Script:

```
using UnityEngine;
using UnityEditor;
using System.Collections;

public class SoundScript : MonoBehaviour {

    float xP,yP,zP,xW,yW,zW,xF,yF,zF,euclW,valueW,euclF,valueF,euclR,valueR,rate;
    int surfaceIndex = 1;
    int velocityW=1;
    bool OnOff=false;
    private Terrain terrain;
    private TerrainData terrainData;
    private Vector3 terrainPos;
    private Collision colision;

    void Start () {
        terrain = Terrain.activeTerrain;
        terrainData = terrain.terrainData;
        terrainPos = terrain.transform.position;
    }

    // Update is called once per frame
    void Update () {
        float xP = transform.position.x;                                    //Posició
n del personaje
        float yP = transform.position.y;
        float zP = transform.position.z;
        float xW = GameObject.Find("Waterfall").transform.position.x;       //Posici
ón de la cascada y distancia euclídea con el personaje
        float yW = GameObject.Find("Waterfall").transform.position.y;       //Usamos
 la componente "x" de la cascada para la distancia al río
        float zW = GameObject.Find("Waterfall").transform.position.z;
        float xF = GameObject.Find("Fire1").transform.position.x;           //Posici
ón de la hoguera y distancia euclídea con el personaje
        float yF = GameObject.Find("Fire1").transform.position.y;
        float zF = GameObject.Find("Fire1").transform.position.z;
        euclW = Mathf.Sqrt((xP - xW)*(xP - xW) + (yP - yW)*(yP - yW) + (zP -
 zW) *(zP - zW));
        euclF = Mathf.Sqrt((xP - xF)*(xP - xF) + (yP - yF)*(yP - yF) + (zP -
 zF) *(zP - zF));
        euclR = Mathf.Sqrt((xP - xW)*(xP - xW));

        if (euclW <= 70) {                                                  //A una
 distancia de 70, máxima amplitud
            valueW = 1;                                                     //A un
a distancia de 410, mínima amplitud (0.03), entra ambas interpolación
        } else {
            if (euclW >= 300) {
                valueW = (float)0.03;
            } else {
                valueW = 1 + (euclW - 70)*((float)0.03-1) / (300 - 70);
            }
        }
        if (euclF <= 2) {                                                   //A una
distancia de 2, máxima amplitud
            valueF = (float)0.2;
   //A una distancia de 410, mínima amplitud (0.03), entra ambas interpolación
        } else {
            if (euclF >= 30) {
                valueF = 0;
```

```
            } else {
                valueF = (float)0.2 + (euclF - 2)*(0-(float)0.2) / (30 - 2);
            }
        }
        if (euclR <= 10) {                                                  //A una
 distancia de 70, máxima amplitud
            valueR = 30;                                                     //A u
na distancia de 410, mínima amplitud (0.03), entra ambas interpolación
        } else {
            if (euclR >= 100) {
                valueR = 0;
            } else {
                valueR = 30 + (euclR - 10)*(0-30) / (100 - 10);
            }
        }
        if ((EditorApplication.isPlayingOrWillChangePlaymode)&(!OnOff)){
            OSCHandler.Instance.SendMessageToClient("PDOnOff", "/state", 1);
            OnOff=true;
        }
        if (!EditorApplication.isPlayingOrWillChangePlaymode){
            OSCHandler.Instance.SendMessageToClient("PDOnOff", "/state", 0);
        }
        OSCHandler.Instance.SendMessageToClient("PDWaterfall", "/waterfall", valueW);
        OSCHandler.Instance.SendMessageToClient("PDFire", "/fire", valueF);
        OSCHandler.Instance.SendMessageToClient("PDWind", "/windon", 1);
        OSCHandler.Instance.SendMessageToClient("PDWind", "/velocity", velocityW);
        if (Input.GetKeyDown (KeyCode.V)) {
            if (velocityW == 1) {
                velocityW = 2;
            } else {
                velocityW = 1;
            }
        }
        OSCHandler.Instance.SendMessageToClient("PDRiver", "/river", valueR);
    }

    void WalkSound () {
              //Controla el sonido con el personaje caminando
        rate = 200;
        if (surfaceIndex != 2) {
            surfaceIndex = GetMainTexture (transform.position);
               //1-Césped, 2-Madera, 3-Nieve, 4-Gravilla
            if (surfaceIndex == 2)
                surfaceIndex = 1;
        }
        if ((surfaceIndex == 0) | (surfaceIndex == 5))
            surfaceIndex = 1;
        Debug.Log ("Superficie: " + surfaceIndex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/surface", surfaceInd
ex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/rate", rate);
    }

    void RunSound () {
             //Controla el sonido con el personaje caminando
        rate = 500;
        if (surfaceIndex != 2) {
            surfaceIndex = GetMainTexture (transform.position);
               //1-Césped, 2-Madera, 3-Nieve, 4-Gravilla
            if (surfaceIndex == 2)
                surfaceIndex = 1;
        }
        if ((surfaceIndex == 0) | (surfaceIndex == 5))
            surfaceIndex = 1;
        Debug.Log ("Superficie: " + surfaceIndex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/surface", surfaceInd
```

```
ex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/rate", rate);
    }

    void SprintSound () {
                //Controla el sonido con el personaje caminando
        rate = 700;
        if (surfaceIndex != 2) {
            surfaceIndex = GetMainTexture (transform.position);
               //1-Césped, 2-Madera, 3-Nieve, 4-Gravilla
            if (surfaceIndex == 2)
                surfaceIndex = 1;
        }
        if ((surfaceIndex == 0) | (surfaceIndex == 5))
            surfaceIndex = 1;
        Debug.Log ("Superficie: " + surfaceIndex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/surface", surfaceInd
ex);
        OSCHandler.Instance.SendMessageToClient("PDFootsteps", "/rate", rate);
    }

    void StopSound (){
        OSCHandler.Instance.SendMessageToClient ("PDFootsteps", "/rate", 0);
    }


    private float[] GetTextureMix(Vector3 WorldPos){
        // returns an array containing the relative mix of textures
        // on the main terrain at this world position.

        // The number of values in the array will equal the number
        // of textures added to the terrain.

        // calculate which splat map cell the worldPos falls within (ignoring y)
        int mapX = (int)(((WorldPos.x -
 terrainPos.x) / terrainData.size.x) * terrainData.alphamapWidth);
        int mapZ = (int)(((WorldPos.z -
 terrainPos.z) / terrainData.size.z) * terrainData.alphamapHeight);

        // get the splat data for this cell as a 1x1xN 3d array (where N = number of
textures)
        float[,,] splatmapData = terrainData.GetAlphamaps( mapX, mapZ, 1, 1 );

        // extract the 3D array data to a 1D array:
        float[] cellMix = new float[ splatmapData.GetUpperBound(2) + 1 ];

        for(int n=0; n<cellMix.Length; n++){
            cellMix[n] = splatmapData[ 0, 0, n ];
        }
        return cellMix;
    }

    private int GetMainTexture(Vector3 WorldPos){
        // returns the zero-based index of the most dominant texture
        // on the main terrain at this world position.
        float[] mix = GetTextureMix(WorldPos);

        float maxMix = 0;
        int maxIndex = 0;

        // loop through each mix value and find the maximum
        for(int n=0; n<mix.Length; n++){
            if ( mix[n] > maxMix ){
                maxIndex = n;
                maxMix = mix[n];
            }
```

```
        }
        return maxIndex;
    }

    void OnTriggerEnter(Collider other) {
        surfaceIndex=2;
    }
    void OnTriggerStay(Collider other) {
        surfaceIndex=2;
        Debug.Log ("On bridge");
    }
    void OnTriggerExit(Collider other) {
        surfaceIndex=1;
    }
}
```

Sound Script se encarga de configurar los parámetros de la mayoría de los efectos de sonido, a excepción del trueno y la lluvia. El código se divide en la declaración inicial de las variables para la clase y en diversas funciones.

La función Start() se ejecuta una vez al inicio de la ejecución de la escena, sirve para obtener los datos del terreno que se utilizará en la posterior función que determina en qué superficie se encuentra el personaje.

La función Update() se ejecuta con cada nuevo “frame”, esto significa que se encuentra en constante ejecución. Se encarga de obtener la posición del personaje en el mapa y la de los diversos elementos que influyen en los efectos de sonido (cascada, hoguera y río), establece unos umbrales a partir de los cuales los sonidos están activos y calcula las distancias euclídeas con las que calcula los parámetros de amplitud para estos efectos. Finalmente, envía vía mensajes OSC estos parámetros a los puertos de escucha

Las funciones WalkSound(), RunSound() y SprintSound() están configuradas para ejecutarse tras eventos procedentes de las animaciones. Cuando se está ejecutando la animación de caminar, trotar o esprintar y el personaje toca el suelo, un evento provoca que se ejecute la función correspondiente (WalkSound() para caminar, RunSound() para trotar, SprintSound() para esprintar). Las tres funciones difieren únicamente en el valor que le dan al parámetro /rate.

La función StopSound() se activa por eventos, al igual que las anteriores, pero se ejecuta en el momento en que el personaje levanta el pie del suelo, poniendo el parámetro /rate a 0 y parando la ejecución del efecto de las pisadas. Esta solución es implementada de esta manera puesto que sería muy complicado sincronizar la animación con un efecto de pisadas ejecutado continuamente. De esta manera, se han elegido tres valores aproximados para el parámetro /rate para cada uno de los tres ritmos y, cuando el personaje levanta el pie, el efecto se apaga, para volver a activarse en la siguiente pisada.

Las funciones GetTextureMix() y GetMainTexture() son utilizadas para averiguar la superficie sobre la que está el personaje en las funciones WalkSound(), RunSound() y SprintSound(). Han sido implementadas por el usuario “alucardj” en el foro de formación sobre Unity 3D “answers.unity3d.com” [36].

Por último, las funciones OnTriggerEnter(Collider other), OnTriggerStay(Collider other) y OnTriggerExit(Collider other) sirven, respectivamente, para saber cuándo el personaje entra,

está sobre y sale de los diversos puentes de madera esparcidos por el mapa. De esta manera es posible darle al parámetro /surface del efecto de pisadas el valor correspondiente a la madera. La variable de entrada “Collider” es activada por un componente del objeto “puente” cuando el personaje entra en contacto con uno de ellos.

## Hero Script:

```
using UnityEngine;
using System.Collections;

public class HeroScript : MonoBehaviour {

    Animator anim;
    int magicHash = Animator.StringToHash("Magia"); //Pasa el string a tipo "Hash ID"
 para no evaluarla en cada ejecución y aumentar la eficiencia
    int leftHash = Animator.StringToHash("Left");
    int rightHash = Animator.StringToHash("Right");

    void Start () {
        anim = GetComponent<Animator> (); //Toma el Animator
    }

    void Update () {
        float move = Input.GetAxis ("Vertical"); //Asigna a la variable "move" la ent
rada del eje en "Vertical"
        if (Input.GetKey (KeyCode.C)) {
            move = move / 2;
        }
        if (Input.GetKey (KeyCode.F)) {            //Caminar con C pulsada, correr má
s rápido con F pulsada
            move = move*2;
        }
        anim.SetFloat ("Speed", move); //Pasa el valor de "move" al parámetro "Speed"
 del Animator

        if (Input.GetKeyDown (KeyCode.M)) {
            anim.SetTrigger (magicHash);
        }

        if (Input.GetKeyDown (KeyCode.T)) {
            anim.SetTrigger (magicHash);
        }

        if (Input.GetKeyDown (KeyCode.R)) {
            anim.SetTrigger (magicHash);
        }

        if (Input.GetKeyDown (KeyCode.A)) {
            anim.SetTrigger (leftHash);
        }

        if (Input.GetKeyDown (KeyCode.D)) {
            anim.SetTrigger (rightHash);
        }

        if (Input.GetKeyDown (KeyCode.V)) {
            anim.SetTrigger (magicHash);
        }
    }
}
```

Este código es utilizado para definir las teclas que controlan al personaje y cómo influyen en la máquina de estados de las animaciones de la Figura (). En la función Start(), que se ejecuta al inicio una única vez, ponemos en una variable “anim” las características de la máquina de estados (Animator). En la función Update(), ejecutada en cada “frame”, definimos una serie de bloques condicionales que cambias las variables que influyen en dicha máquina de estados, en consonancia con las teclas que pulsemos. De esta manera, “GetKeyDown” detecta simplemente cuando la tecla es pulsada, mientras que “GetKey” detecta cuando la tecla es pulsada y se mantiene pulsada. Por ello, “GetKey” se utiliza para el avance del personaje y el ritmo y “GetKeyDown” para los giros, truenos, lluvia y cambio de velocidad del viento.

## Rain Script

```
using UnityEngine;
using System.Collections;
using UnityOSC;

public class RainScript : MonoBehaviour {

    public GameObject myObject;
    void Start () {
        OSCHandler.Instance.Init ();
        myObject.SetActive(false);
        OSCHandler.Instance.SendMessageToClient("PDRain", "/rainoff", 1);
    }

    void Update () {
        if (Input.GetKeyDown (KeyCode.R)) {
            if (myObject.activeSelf) {
                myObject.SetActive (false);
                OSCHandler.Instance.SendMessageToClient("PDRain", "/rainoff", 1);
            }
            else {
                myObject.SetActive (true);
                OSCHandler.Instance.SendMessageToClient("PDRain", "/rainon", 1);
            }
    }
}
}
```

Este “script” es utilizado para controlar el sonido de la lluvia. En la función Start() se pone el objeto que representa la lluvia en la escena desactivado y se dice a PD que apague el efecto del sonido. En la función Update(), el programa espera a la pulsación de la tecla “R” para actuar. Alternativamente cambiará el estado del objeto en la escena y del efecto en PD con cada pulsación.

## Flash

```
using UnityEngine;
using System.Collections;

public class flash : MonoBehaviour {
    private Light myLight;
    int countdown;
    void Start (){
        myLight = GetComponent<Light> ();
        countdown = 50;
        light.enabled = false;
    }

    void Update () {
        if (Input.GetKeyDown (KeyCode.T)) {
            light.enabled = true;
            countdown = 50;
            OSCHandler.Instance.SendMessageToClient("PDThunder", "/thunder", 1);
        } else {
            countdown--;
            if(countdown<=0){
                light.enabled = false;
            }
        }
    }

}
```

Este “script” se encarga de reproducir visual y acústicamente el efecto del trueno. En la función Start() se ponen en la variable “myLight” las características del objeto que se encargará de iluminar la escena, simulando el efecto de un rayo, cuando vaya a sonar un trueno. La variable “countdown” representa una cuenta atrás que irá decreciendo con cada “frame”, al final de la cual el objeto dejará de iluminar la escena. En la función Update(), se espera a la pulsación de la tecla “T” para iluminar la escena, empezar la cuenta atrás y activar el efecto de sonido.

## OSCHandler

```
public void Init()
    {
        CreateClient("PDRain", IPAddress.Parse("127.0.0.1"), 7001);
        CreateClient("PDThunder", IPAddress.Parse("127.0.0.1"), 7002);
        CreateClient("PDWaterfall", IPAddress.Parse("127.0.0.1"), 7003);
        CreateClient("PDWind", IPAddress.Parse("127.0.0.1"), 7004);
        CreateClient("PDFire", IPAddress.Parse("127.0.0.1"), 7005);
        CreateClient("PDRiver", IPAddress.Parse("127.0.0.1"), 7006);
        CreateClient("PDFootsteps", IPAddress.Parse("127.0.0.1"), 7007);
        CreateClient("PDOnOff", IPAddress.Parse("127.0.0.1"), 7008);
    }
```

En el código llamado “OSCHandler” de la librería “UnityOSC-master” de Jorge García [35] se incluye esta función Init() que se utiliza para inicializar con quién ejerce Unity de cliente (en este caso, PD) y los servidores si hubieran otros programas que enviaran información a Unity.

# Anexo B: Scripts en Matlab para la obtención de las Figuras

## Pisadas sobre nieve: envelopeSnow.m

```
%%%%%Lee y representa el valor absoluto de las pisadas en el
tiempo%%%%
[pisadas,Fs] = wavread('FootstepsSnow.wav');
N = length(pisadas);
t = linspace(0, N/Fs, N);
figure(1), plot(t,pisadas(:,1)); title('Pisadas'); xlabel('Tiempo');
ylabel('Señal');

%%%%%Calcula y representa la FFT de una pisada%%%%%
pisa = pisadas(5.8*Fs:(65*Fs/10),1);
Np = length(pisa);
tp=linspace(0,Np/Fs,Np);
figure(2),plot(tp,pisa); title('Pisada en tiempo');
PISA=fft(pisa);
f = Fs/2*linspace(0,1,Np/2+1);
figure(3),plot(f,abs(PISA(1:length(f)))); title('Pisada en
frecuencia'); xlabel('Frecuencia'); ylabel('Señal');

%%%%%Calcula y representa la envolvente temporal%%%%%
pisadas2 = 2*pisadas.*pisadas;
downsample(pisadas2,15);
Fc=1;
[B,A] = butter(1,Fc/(Fs/15));
pisadas2 = sqrt(filter(B,A,pisadas2));
figure(4), plot(t,pisadas2(:,1)); title('Envolventes');
xlabel('Tiempo')
```

## Pisadas sobre gravilla: envelopeGravel.m

```
%%%%%Lee y representa las pisadas en el tiempo%%%%
[pisadas,Fs] = wavread('FootstepsGravel');
N = length(pisadas);
t = linspace(0, N/Fs, N);
figure(1), plot(t,pisadas); title('Pisadas');

%%%%%Calcula y representa la FFT de una pisada%%%%%
pisa = pisadas(7.7*Fs:(82*Fs/10-1),:);
Np = length(pisa);
tp=linspace(0,Np/Fs,Np);
figure(2),plot(tp,pisa); title('Pisada en tiempo'); xlabel('Tiempo');
ylabel('Señal');
PISA=fft(pisa);
f = Fs/2*linspace(0,1,Np/2+1);
figure(3),plot(f,abs(PISA(1:length(f)))); title('Pisada en
frecuencia'); xlabel('Frecuencia'); ylabel('Señal');
```

## Pisadas sobre madera: envelopeWood.m

```
%%%%%Lee y representa las pisadas en el tiempo%%%%
[pisadas,Fs] = wavread('FootstepsWood.wav');
N = length(pisadas);
t = linspace(0, N/Fs, N);
figure(1), plot(t,pisadas); title('Pisadas');

%%%%%Calcula y representa la FFT de varias pisadas%%%%%
pisa1 = pisadas(3.4*Fs:(39*Fs/10-1),:);
Np1 = length(pisa1);
tp1=linspace(0,Np1/Fs,Np1);
figure(2), plot(tp1,pisa1);
PISA1=fft(pisa1);
f1 = Fs/2*linspace(0,1,Np1/2+1);
figure(3),plot(f1,abs(PISA1(1:length(f1)))); title('Pisada en
frecuencia 1'); xlabel('Frecuencia'); ylabel('Señal');

pisa2 = pisadas(4.2*Fs:(47*Fs/10-1),:);
Np2 = length(pisa2);
tp2=linspace(0,Np2/Fs,Np2);
PISA2=fft(pisa2);
f2 = Fs/2*linspace(0,1,Np2/2+1);
figure(4),plot(f2,abs(PISA2(1:length(f2)))); title('Pisada en
frecuencia 2');


%%%%%Calcula y representa la envolvente temporal%%%%%
pisadas2 = 2*pisadas.*pisadas;
downsample(pisadas2,15);
Fc=1;
[B,A] = butter(1,Fc/(Fs/15));
pisadas2 = sqrt(filter(B,A,pisadas2));
figure(6), plot(t,pisadas2(:,1)); xlabel('Tiempo'); ylabel('Señal');
```

## Pisadas sobre follaje: envelopeGrass.m

```
%%%%%Lee y representa las pisadas en el tiempo%%%%
[pisadas,Fs] = wavread('FootstepsGrass.wav');
N = length(pisadas);
t = linspace(0, N/Fs, N);
figure(1), plot(t,pisadas); title('Pisadas');

%%%%%Calcula y representa la FFT de varias pisadas%%%%%
pisa2 = pisadas(0.7*Fs:(11*Fs/10),1);
Np2 = length(pisa2);
tp=linspace(0,Np2/Fs,Np2);
figure(2),plot(tp,pisa2); title('Pisada2 en tiempo');
PISA2=fft(pisa2);
f2 = Fs/2*linspace(0,1,Np2/2+1);
figure(3),plot(f2,abs(PISA2(1:length(f2)))); title('Pisada en
frecuencia');xlabel('Frecuencia');ylabel('Señal');

pisa = pisadas(0.1*Fs:(6*Fs/10),1);
Np = length(pisa);
PISA=fft(pisa);
```

```
f = Fs/2*linspace(0,1,Np/2+1);
figure(2),plot(f,abs(PISA(1:length(f)))); title('Pisada1 en
frecuencia');

pisa3 = pisadas(1.2*Fs:(17*Fs/10),1);
Np3 = length(pisa3);
PISA3=fft(pisa3);
f3 = Fs/2*linspace(0,1,Np3/2+1);
figure(4),plot(f3,abs(PISA3(1:length(f3)))); title('Pisada3 en
frecuencia');

pisa4 = pisadas(1.75*Fs:(22*Fs/10),1);
Np4 = length(pisa4);
PISA4=fft(pisa4);
f4 = Fs/2*linspace(0,1,Np4/2+1);
figure(5),plot(f4,abs(PISA4(1:length(f4)))); title('Pisada4 en
frecuencia');

%%%%%Calcula y representa la envolvente temporal%%%%%
pisadas2 = 2*pisadas.*pisadas;
downsample(pisadas2,15);
Fc=1;
[B,A] = butter(1,Fc/(Fs/15));
pisadas2 = sqrt(filter(B,A,pisadas2));
figure(6), plot(t,pisadas2(:,1));
title('Envolventes');xlabel('Tiempo');ylabel('Señal');
```

## Río: river.m

```
%%%%%Representa la señal en el tiempo%%%%%
[rio,Fs]=wavread('rio.wav');
N = length(rio);
t = linspace(0, N/Fs, N);
figure(1), plot(t,rio(:,1)); title('Río (tiempo)'); xlabel('Tiempo');
ylabel('Señal');


%%%%%Calcula y representa la Transformada de Fourier de la señal%%%%%
RIO = fft(rio);
f = Fs/2*linspace(0,1,N/2+1);
figure(2),plot(f,abs(RIO(1:length(f)))); title('Río (frecuencia)');
xlabel('Frecuencia'); ylabel('Señal');
```

## Cascada: waterfall.m

```
%%%%%Representa la señal en el tiempo%%%%%
[waterfall,Fs]=wavread('waterfall.wav');
N = length(waterfall);
t = linspace(0, N/Fs, N);
figure(1), plot(t,waterfall(:,1)); title('Cascada (tiempo)');
xlabel('Tiempo'); ylabel('Señal');

%%%%%Calcula y representa la Transformada de Fourier de la señal%%%%%
WATERFALL = fft(waterfall);
f = Fs/2*linspace(0,1,N/2+1);
figure(2),plot(f,abs(WATERFALL(1:length(f)))); title('Cascada
(frecuencia)'); xlabel('Frecuencia'); ylabel('Señal');
```

# Anexo C: Referencias


[1] Farnell, A.J. (2007) *An introduction to procedural audio and its application in computer games*

[2] Cook, P.R. (2001) *Modelling Bill's Gait: Analysis and parametric synthesis of walking sounds*

[3] Bresin, R., Friberg, A. y Dahl, S. (2001) *Toward a new model of sound control*

[4] Bresin, R. y Fontana, F. (2003) *Physic-Based sound synthesis and control: crushing, walking and running by crumpling sounds*

[5] Pathon, C. (2011) *Modelling footsteps: Procedural Audio in games*

[6] Zheng, C. y James D.L. (2006) *Harmonic fluids*

[7] Milavcic, J.M., Zita, A. y Arvidsson, P. (2004) *Computational real-time sound synthesis of rain*

[8] van den Doel, K. (2004) *Physically-based models for liquid sounds*

[9] Moss, W., Yeh, H., Hong, J., Ming, C.L. y Manocha, D. (2010) *Sounding liquids: Automatic sound synthesis from fluid simulation*

[10] Dobashi, Y., Yamamoto, T. y Tomoyuki, N., (2004) *Synthesizing sound from turbulent field using sound textures for interactive fluid simulation*

[11] Chadwick, J.N. y James, D.L. (2011) *Animating fire with sound*

[12] Verron, C. y Drettakis, G. (2012) *Procedural audio modelling for particle-based environmental effects*

[13] Fagerlund, S. (2004) *Acoustics and physical models of bird sounds*

[14] Kahrs, M. y Avanzini, F. (2001) *Computer synthesis of bird songs and calls*

[15] Smyth, T. y Smith III, J.O. (2002) *The sounds of the avian syrinx – are they really flute-like?*

[16] Smyth, T., Abel, J.S. y Smith III, J.O. (2003) *The estimation of birdsong control parameters using maximum likelihood and minimum action*

[17] Brothers, D.J. y Tschuch, G. (1999) *Modelling vibration and sound production in insects with nonresonant stridulatory organs*

[18] Smyth, T. y Smith III, J.O. (2001) *Applications of bioachoustics in physical modelling and the creation of new musical instruments*

[19] Cook, P. (1991) *Identification of control parameters in an articulary vocal tract model, with applications to the synthesis of singing*

[20] Martino, R. (2000) *Synthasaurus: an animal vocalization synthesizer*

[21] van den Doel, K., Kry, P.G. y Pai, D.K. (2002) *Physically-based sound ejects for interactive simulation and animation*

[22] van den Doel, K. y Pai, D.K. (2002) *Modelling synthesis for vibrating objects*

[23] O'Brien, J.F., Shen, C. y Gatchalian, C.M. (2002) *Synthesizing sounds from rigid body simulations*

[24] James, D.L., Barbic, J. y Pai, D.K. (2007) *Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources*

[25] Rath, M. Avanzini, F., Bernardini, N., Borin, G., Fontana, F., Ottaviani, L., Rochesso, D. (2003) *An introductory catalog of computer-synthesized contact sounds, in real-time*

[26] Bonneel, N., Drettakis, G., Tsingos, N., Viaud-Delmon, I. y James, D.(2007) *Fast modal sounds with scalable frequency-domain synthesis*

[27] Zheng, C. y James, D.L. (2011) *Toward high-quality modal contact sound*

[28] Farnell, A. (2007) *Synthetic game audio with Puredata*

[29] Paul, L.J. (2007) *Video game audio prototyping with Pure Data*

[30] Alonso, M., Geiger, G., y Jorda, S. (2004) *An internet browser plug-in for real-time audio synthesis. WEDELMUSIC '04*

[31] Paul, L.J. (2003) *Audio prototyping with Pure Data*

[32] http://docs.unity3d.com/Manual/UnityOverview.html

[33] Farnell, A. (2010) *Designing sound*

[34] Wright, M (2002) *OpenSound Control specification*

[35] Librería UnityOSC-master de Jorge García, 2012: *https://github.com/jorgegarcia/UnityOSC/tree/master/src/OSC*

[36] Funciones de "alucardj": *http://answers.unity3d.com/questions/456973/getting-the-texture-of-a-certain-point-on-terrain.html*

[37] www.mathworks.es

[38] https://creative.adobe.com